2ème EDITION



ETUDES AUTOUR DU 6809

(CONSTRUCTIONS ET LOGICIELS)





Claude Vicidomini



ETUDES AUTOUR DU 6809

(CONSTRUCTIONS ET LOGICIELS)

2e édition



Diffusé par EYROLLES
61, bd Saint-Germain, 75240 Paris Cédex 05

Société éditrice : Editions Fréquences Siège Social : 1, bd Ney, 75018 Paris - Tél. : (1) 40.36.01.97 + - SA au capital de 1.000.000 F - Président-Directeur Général : Edouard Pastor.
LED Initiation - Etudes autour du 6809 - 140 F - Commission paritaire : 64949 - Directeur de la publication : Edouard Pastor - Tous droits de reproduction et d'adaptation réservés textes et photos pour tous pays - LED est une marque déposée ISSN 0753-7409.
Rédaction : Claude Vicidomini.
Photocomposition, mise en page, photogravure et réalisation : Edi'Systèmes.
Impression : Berger-Levrault - Nancy.

Chapitre 1

De la logique câblée au microprocesseur

Introduction au microprocesseur

L'être physique : l'homme

Le génie créatif de l'homme est probablement sans limite ; voici qu'il a inventé, il y a maintenant plus de dix années, le microprocesseur qui nous ouvre toutes grandes les portes de l'ère post-industrielle.
Le microprocesseur est ce que l'on pourrait appeler le «cerveau profond» d'une machine intelligente.

Notion d'environnement

Nous sommes en permanence baignés dans un environnement qui nous communique, par l'intermédiaire d'organes, le contexte dans lequel nous nous trouvons.
Que nous soyons endormis ou éveillés, notre cerveau effectue un scanning permanent. Il est toujours en éveil mais il nous cachera certains résultats, jugeant qu'ils ne sont pas assez importants pour être mis au niveau de notre conscience.
Par exemple : la station debout exige des efforts permanents des muscles de notre corps pour garder l'équilibre. Notre oreille interne fournit au cerveau le sens de la pesanteur. Si nos jambes ressentent le poids de notre corps, nous sommes donc debout ; si notre postérieur ressent ce poids, nous sommes assis.
En fait, notre cerveau effectue des opérations logiques. Dans l'exemple ci-dessus, il s'agit d'un ET, les notions de verticalité et de poids s'appellent des paramètres, ainsi Station debout (d) = Vertical (v) ET Poids (p) ou d = v.p.
Il est bien évident que l'association de plusieurs paramètres (plus de 2) nous place dans un certain contexte.
Il peut y avoir contradiction et, dans ce cas, nous ne savons plus où nous en sommes, ce qui entraîne, chez l'homme, des réactions tout à fait semblables à un malaise.
Par exemple, le fait de se trouver dans l'espace, sans poids et sans référence de pesanteur amène notre cerveau à prendre de nouvelles habitudes... mais que la pesanteur revienne brutalement et c'est le malaise qui entraîne des vomissements : le cerveau ne sait plus quels sont les bons paramètres à prendre en compte ! Autrement dit : d ≠ v.p.
L'exemple que nous venons de citer constitue une expérience très importante de Patrick Baudry dans le vol orbital de la navette spaciale.
D'autres contextes entraînent d'autres formulations logiques : on peut introduire le OU logique (la saveur de ce met me laisse à penser qu'on a un peu forcé sur l'épice OU que le plat est naturellement épicé OU les deux à la fois), le OU exclusif (uniquement l'un OU uniquement l'autre). Toutes ces formulations de type booléenne peuvent être fort complexes et notre cerveau est parfaitement capable de trouver la ou les racines de toute équation.

Notion d'organes

Pour que notre cerveau soit capable de sûpputer et de prendre des décisions pour un contexte donné, il faut le brancher à des organes capables de recueillir ou d'envoyer toutes sortes d'informations..., et elles sont variées :
– l'intensité lumineuse est une première information : elle se cantonne dans le cas de l'homme, dans le spectre visible, l'œil est l'organe essentiel de la vue. Notre peau, dans une moindre mesure, capte des rayonnements infra-rouges nous donnant ainsi la sensation du chaud et du froid.
– le bruit, interprété ici dans sa forme générale (des vibrations de fréquence très basse: infrasons ou ultrasons), est un support extraordinaire qui nous permet de comprendre (l'ouïe) et de nous faire comprendre (la voix). Ce bruit peut être harmonieux (le chant) ou franchement désagréable (le son d'une guitare désaccordée ou les réacteurs d'un avion en phase de décollage). Notons que dans ce dernier cas, notre corps se met en vibration en résonance avec les infrasons engendrés par les réacteurs).
– le volume d'un objet nous est communiqué à la fois par nos mains et notre vue : nous savons reconnaître une cuillère ou une fourchette les yeux fermés, nous savons évaluer le poids de ce tome ainsi que ses dimensions toujours les yeux fermés, mais nous n'aurons jamais l'idée de faire de même avec un immeuble de 50 étages !
C'est que là, notre cerveau associé à notre vue, est capable de faire des prodiges d'ingéniosité : puisqu'il est impossible de soupeser cet immeuble, vu ses dimensions, il peut aller chercher en mémoire une information relative à un immeuble d'une taille semblable lue dans un article d'une revue puis négocier le poids réel en fonction des informations recueillies par la vue.
Une autre solution consiste à calculer la masse de l'immeuble en évaluant le poids d'une quantité de maçonnerie connue et en le multipliant par le nombre d'éléments constituant l'immeuble.
D'autres solutions existent encore et l'imagination de notre cerveau est sans limite pour faire face à de telles situations.

Notion de mémoire

Nous l'avons vu dans l'exemple ci-dessus, une mémoire est indispensable pour nous permettre de nous souvenir d'événements ayant un rapport avec la situation que nous vivons.

Notre cerveau dispose de plusieurs milliards de ces boîtes aux lettres qui s'enrichissent sans cesse de faits nouveaux. En fait, cette mémoire «boîte aux lettres» nous sert à évaluer, comparer, négocier une situation par rapport à des paramètres préétablis ; considérons l'expérience suivante :
– dans une boîte, disposons 8 boules de couleur noire et 2 boules de couleur blanche, puis fermons le couvercle et, en agitant bien l'ensemble, faisons tomber 5 boules dans une seconde boîte. Le contenu de la deuxième boîte nous donnera aussitôt le contenu de la première par simple déduction logique.
Si nous y trouvons 5 boules noires, il reste donc dans la première boîte 3 boules noires et 2 blanches.
Si nous trouvons 2 boules blanches dans la deuxième boîte, il reste donc 5 boules noires dans la première boîte...
Les données de départ étaient :
Paramètre 1 = 2 boules blanches
Paramètre 2 = 8 boules noires
Total = 10 boules = paramètre 1 + paramètre 2
Les données d'arrivées sont donc :
$0 <=$ Paramètre 1 $<= 2$
$0 <=$ Paramètre 2 $<= 5$
Notre cerveau effectue donc une simple opération arithmétique amenant le résultat final mais il fallait bien qu'il demande à notre mémoire «boîte aux lettres» les paramètres de départ pour en tirer la conclusion qui s'impose.

Notion d'ordonnancement ou de séquencement

Nous en arrivons au point final et crucial du fonctionnement d'une machine humaine : toutes les données fournies par notre environnement ainsi que toutes celles que nous lui fournissons doivent être «temporisées». Une unité de séquencement se chargera de l'affaire.
Sa présence est indispensable car sans elle tout serait mélangé, sans queue ni tête ; bref, ce serait la confusion totale !
Imaginez-vous donc en train de vous servir un verre d'eau... sans verre, ou prendre votre soupe avec une fourchette... les exemples abondent, il suffit d'ailleurs de citer le cas des fous qui ne peuvent plus négocier leurs actes d'une façon logique.

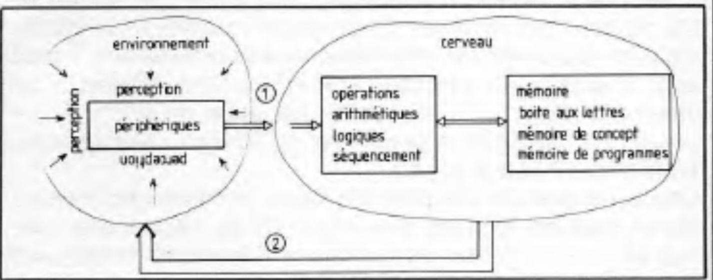


Fig. 1 : Trajet 1 : perception de phénomènes extérieurs. Trajet 2 : action s'il y a lieu.

Résumé

Une machine intelligente, en l'occurrence un être humain, dispose d'un cerveau comprenant :
– une unité arithmétique et logique effectuant toutes les opérations logiques (OU, ET, NON...) et arithmétiques (+, −, ×, ÷...)
– une mémoire «mémorisant» des données (exemple des boules, exemple de l'immeuble) ou exécutant des tâches selon un algorithme préétabli (exemple : la station debout exige l'action de certains muscles du corps pour ne pas perdre l'équilibre...)
– des organes périphériques (mains, yeux, nez...) permettant de communiquer avec le monde extérieur.
– une unité de séquencement (ou d'ordonnancement) synchronisant des tâches élémentaires par rapport à une unité de temps afin de ne pas provoquer de mélanges ou «patchworks» intempestifs.
En réalité, la puissance de notre cerveau permet d'exécuter plusieurs tâches différentes simultanément (le fait de marcher dans la rue ne nous empêche pas de dialoguer avec notre voisin).

L'être physique : le robot

On conçoit aisément que le but à atteindre par tout industriel est de réaliser la machine idéale et parfaite.
Le robot idéal sera ce que les auteurs de science fiction appellent un «androïde» autrement dit un être humain fabriqué industriellement par des humains !
Nous n'en sommes pas encore là car l'évolution de l'informatique en est encore à son balbutiement.
En effet, la difficulté de réalisation d'un robot ne réside pas dans sa partie matérielle : on sait fabriquer des mains ayant autant de degrés de liberté que celles de l'être humain, les yeux remplacés ici par des caméras CCD sont plus sensibles que les yeux humains mais leur définition est bien moindre. Il va sans dire que la technologie évolue à grande vitesse dans ce domaine-là et nous verrons bientôt l'arrivée sur le marché de caméras très performantes ; la synthèse et la reconnaissance vocale font actuellement des progrès extraordinaires.
En fait, la principale difficulté réside dans la réalisation du cerveau de ce robot. Il doit être intelligent, autrement dit, il doit être aussi complexe que le cerveau humain, or, il n'existe actuellement aucune technologie opérationnelle capable de concurrencer notre cerveau. Un deuxième problème, et non le moindre, réside dans l'intelligence de ce cerveau : aucun programme, aucun algorithme ne peut simuler le fonctionnement de notre cerveau.
En fait, ces programmes appelés «systèmes experts» en sont, à leurs tous premiers balbutiements, les industries nipponne et américaine se sont lancées dans la réalisation de programmes experts ouvrant grande la porte de l'intelligence artificielle (notons que l'Europe, un peu à la traîne, songe à combler rapidement ce retard).
Les ordinateurs des années 90 auront peu évolué technologiquement mais les logiciels seront très performants car ils seront intelligents, c'est-à-dire capables d'exécuter les ordres du programmeur avec très peu de manipulations.
Nous avons eu l'occasion d'utiliser un programme d'éducation assisté par ordinateur qui utilise un système expert en guise de logiciel et nous avons été surpris de la rapidité et de la simplicité de mise en œuvre du cours que nous voulions mettre au point.
Le plus grand progrès de cette décennie passera donc par cette voie : imaginez en effet qu'il vous sera enfin possible de «jouer» avec votre ordinateur sans avoir besoin de lire

préalablement le mode d'emploi rédigé en trois tomes !
De plus, tous les programmes qui seront vendus sur le marché pourront être exécutés sur votre ordinateur de même que tous vos programmes pourront être exécutés sur un ordinateur de marque concurrente.
Aujourd'hui, nous ne savons pas fabriquer des robots parfaits, mais nous savons par contre fabriquer des éléments disparates dont l'ensemble constituera une machine intelligente appelée «micro-ordinateur».
La principale différence entre ordinateur et micro-ordinateur réside en 2 points :
– La taille : un ordinateur occupe une surface au sol de plusieurs mètres carrés : il utilise une mémoire très importante et peut se raccorder sur plusieurs ordinateurs formant ainsi un réseau pouvant être très complexe. Plusieurs personnes peuvent faire exécuter des tâches différentes sur cet ordinateur sans qu'il soit trop encombré.
En revanche, un micro-ordinateur occupe peu de place (une petite table lui suffit), sa mémoire est réduite bien qu'elle soit capable d'être agrandie de façon notable, il lui sera difficile de former un réseau mais il peut devenir la liaison terminale d'un ordinateur ; plusieurs personnes peuvent exécuter des tâches différentes mais ce nombre de personnes doit être réduit (de 4 à 16) car sa vitesse de travail diminue à chaque ajout d'un utilisateur... dans ce cas précis, le temps d'accès à une information peut quelquefois être très long.
– Le prix : un ordinateur est par définition un système professionnel, car son coût est souvent très supérieur à 600 KF.
Pour ce prix, on a un système très fiable avec une assistance rapide en cas de panne.
Un «bon» micro-ordinateur dans la gamme des professionnels se situe dans une fourchette de prix entre 100 KF et 200 KF, ses performances sont voisines de celles d'un ordinateur bas de gamme et de plus il bénéficie d'une assistance technique très efficace. Il a aussi l'avantage d'être modulaire, ce qui lui permet de répondre à tous les besoins de l'utilisateur.

Constitution d'une «entité» intelligente

Nous allons tenter ici de schématiser l'organisation d'un système intelligent, et ceci par l'assemblage de boîtes noires, nous entrerons dans les détails un peu plus loin.

La mémoire

En nous reportant à la figure 1, nous remarquons que pour la partie «cerveau», un dialogue permanent doit pouvoir se faire entre mémoire «boîte aux lettres» et l'unité d'opérations et de séquencements ; or, toutes les informations emmagasinées dans la mémoire doivent être recueillies dans le temps le plus court possible afin d'être traitées en «temps réel». Pour arriver à cette finalité, il suffit d'imaginer que la mémoire est organisée en «étages». A chaque étage se trouve l'information soigneusement rangée qu'il suffit d'aller chercher à condition de savoir qu'elle se trouve là ! Une telle organisation est dite aléatoire parce qu'il n'y a qu'une information par étage et qu'il est ainsi facile de la trouver. Par exemple (fig. 2), à l'étage 5 se trouve Jean, et Françoise à l'étage numéro 8...
Nous proscrirons d'emblée l'organisation séquentielle qui obligerait, pour aller chercher Françoise, de passer par les appartements de Catherine, Vincent, Claude..., quelle perte de temps et quels dérangements pour tout le monde ! (fig. 3).

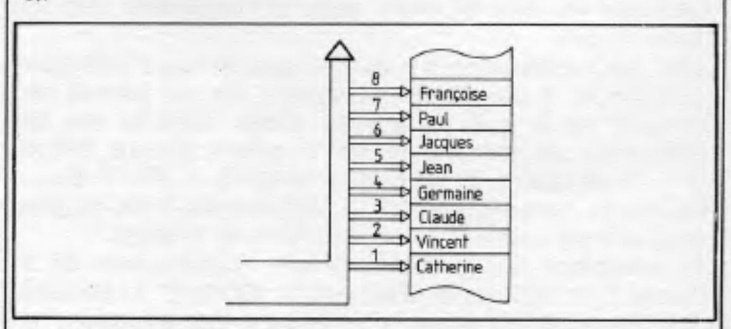


Fig. 2 : Organisation aléatoire de la mémoire.

1	2	3	4	5	6	7	8
Catherine	Vincent	Claude	Germaine	Jean	Jacques	Paul	Françoise

Fig. 3 : Organisation séquentielle de la mémoire.

Afin de bien comprendre ce qui suit, prenons l'exemple suivant :
Nous devons aller chercher Françoise pour aller au cinéma, elle habite le 8[e] étage d'un immeuble un peu vieillot du centre de Montpellier. L'escalier permettant d'accéder aux appartements de l'immeuble est très étroit ; aussi, le syndic des copropriétaires oblige-t-il les personnes à emprunter l'ascenseur pour monter, et l'escalier pour descendre (il est moins essoufflant de descendre un escalier que de le monter).
Une telle organisation permet une plus grande rapidité d'exécution car plusieurs personnes peuvent emprunter l'escalier et/ou l'ascenseur pour des directions différentes ; par contre, si l'ascenseur tombe en panne, nous imaginons sans peine la confusion que cela entraînerait si tout le monde se trouvait en même temps dans l'escalier ; Françoise mettra pas mal de temps pour rejoindre son appartement du 8[e] étage !...

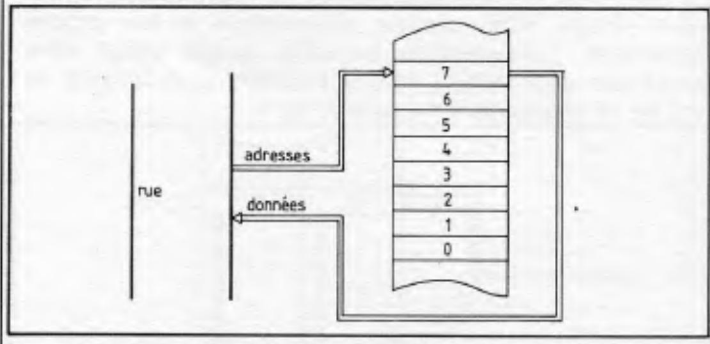


Fig. 4: Structure mémoire standard (on commence toujours par 0).

Nous introduirons ici notre premier terme technique :
L'escalier et l'ascenseur sont des lieux communs que tout le monde peut emprunter : on les appelle «BUS»

L'escalier qui permet d'arriver à l'étage s'appelle un Bus d'Adresses.

L'ascenseur qui récupère le «contenu de l'étage s'appelle Bus de Données.

Une telle architecture est appelée architecture standard : tous les micro-ordinateurs utilisent une telle architecture avec cependant une capacité plus ou moins étendue (l'escalier est plus ou moins large et l'ascenseur plus ou moins grand).

Ainsi, pour notre immeuble de 8 étages, le bus d'adresses comprendra 3 bits (3 fils de liaison), ce qui permet de compter de 0 à 7 (le premier étage étant le rez de chaussée), un bus de 16 fils comptera jusqu'à 65535 ($2^{16}-1$), un bus de 20 fils jusqu'à 1048575 = $2^{20}-1$, etc...

Le bus de données quant à lui, comprendra 8 fils, on dira ainsi qu'il est organisé en octets (Bytes en anglais).

En admettant que Françoise habite l'appartement 85 à l'étage 7, on trouvera la valeur binaire 10000101 à l'adresse 111. On remarque aussi qu'à chacune des adresses, on peut ranger toute valeur comprise entre 0 (00000000) et FF (11111111) soit 0 <= Données <= 255 décimal.

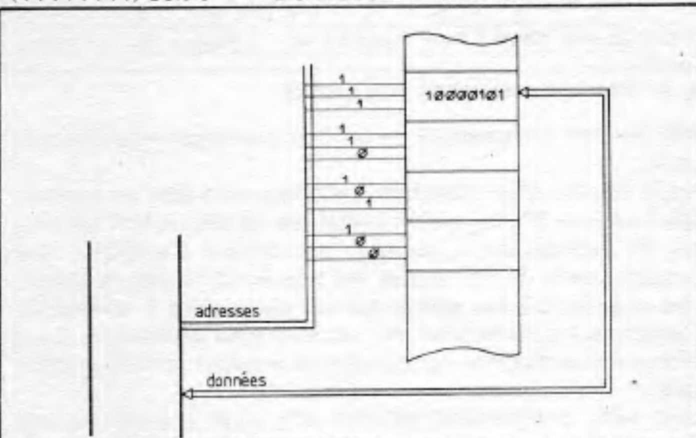


Fig. 5 : Organisation de la mémoire en octets : la capacité de la mémoire correspond à 2^{nbre} de fils d'adresses × 8 = $2^3 \times 8$ = 64 bits.

Il n'est pas bien difficile d'imaginer une structure mémoire organisée en mots (les données sont codées sur 16 bits) : imaginons un building style «Tour Montparnasse» ou «Tour Infernale» pour les ennemis de ce type d'architecture.

La cage d'escalier accède à plusieurs appartements sur le même étage, mais chaque appartement a son propre ascenseur. L'information recueillie à cet étage sera constituée de 2 valeurs dont la concaténation formera un mot de 16 bits (codé sur 2 octets) fig. 6.

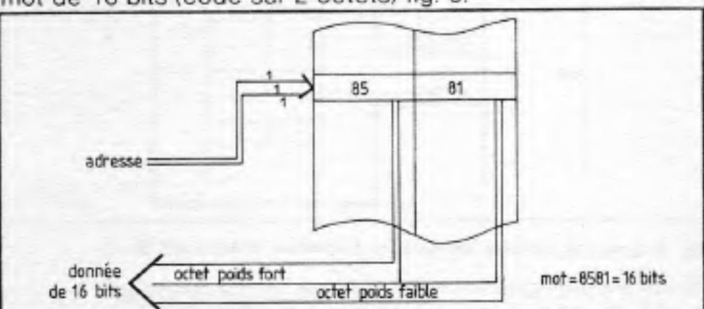


Fig. 6 : Organisation de la mémoire en mots : ici à l'adresse 7 on trouve le mot de 16 bits : 8581 recueilli sur le bus de données.

Il n'y a pas de limitation à l'organisation de la mémoire, elle peut aussi être étendue sur un long mot (32 bits), sur un très long mot (64 bits) etc...

Mais quel casse-tête pour le programmeur, aussi contentons-nous de l'octet !

Enfin, dernière remarque : l'appartement de Françoise pourrait très bien être occupé par Julien ; Françoise prêtant son appartement lorsqu'elle doit s'absenter quelques jours (attention, deux données ne peuvent coexister à la même adresse... il y a dans ce cas un conflit). Cela nécessite d'avoir un bus de données bi-directionnel.

Pour la donnée pouvant être rangée à l'adresse X, l'opération s'appellera une écriture (WRITE) ; pour la donnée pouvant être prise depuis l'adresse X, l'opération s'appellera une lecture (READ) fig. 7.

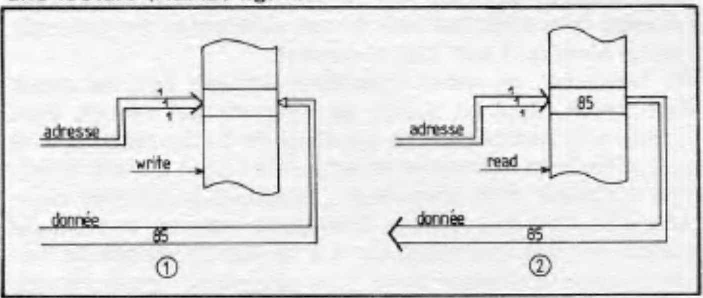


Fig. 7 : Le bus de données est bidirectionnel.

1. Opération d'écriture : le bus de donnée est orienté vers la mémoire grâce à l'ordre «Write».
2. Opération de lecture : le bus de donnée a changé de sens grâce à l'ordre «Read».

L'unité centrale

Nous avons vu précédemment qu'une «unité centrale» était indispensable pour assurer les séquencements des opérations demandées dans un programme. Ces opérations peuvent être de type arithmétique ou logique, il sera judicieux d'intégrer une Unité Arithmétique et Logique (UAL) au sein de l'unité centrale.

Enfin, l'unité centrale ne nous donnera un résultat que dans le cas où elle disposera de tous les renseignements dont elle a besoin ; il faudra donc la munir d'une mémoire bloc-note (scratch pad en anglais), celle-ci étant aussi réduite que possible et se résume souvent à quelques registres.

Prenons le rôle d'un concepteur de systèmes et construisons notre première machine intelligente que nous appellerons MOPET (Micro Ordinateur Pour Etudes Techniques)... et maintenant, laissons notre MOPET faire son show (il fallait bien la placer celle-là) fig. 8.

Avant toute chose, considérons que la mémoire est subdivisée arbitrairement par nous en deux zones : une zone programme qui contiendra notre programme, et une zone données qui contiendra les résultats des opérations demandées par notre programme.

Toute opération demandée à l'unité centrale s'écrira suivant une syntaxe particulière :

– une ligne de programme = (Code Opératoire) + (Opérande) = une instruction

– ou une ligne de programme = (Additionne) + (2 + 2) = une instruction

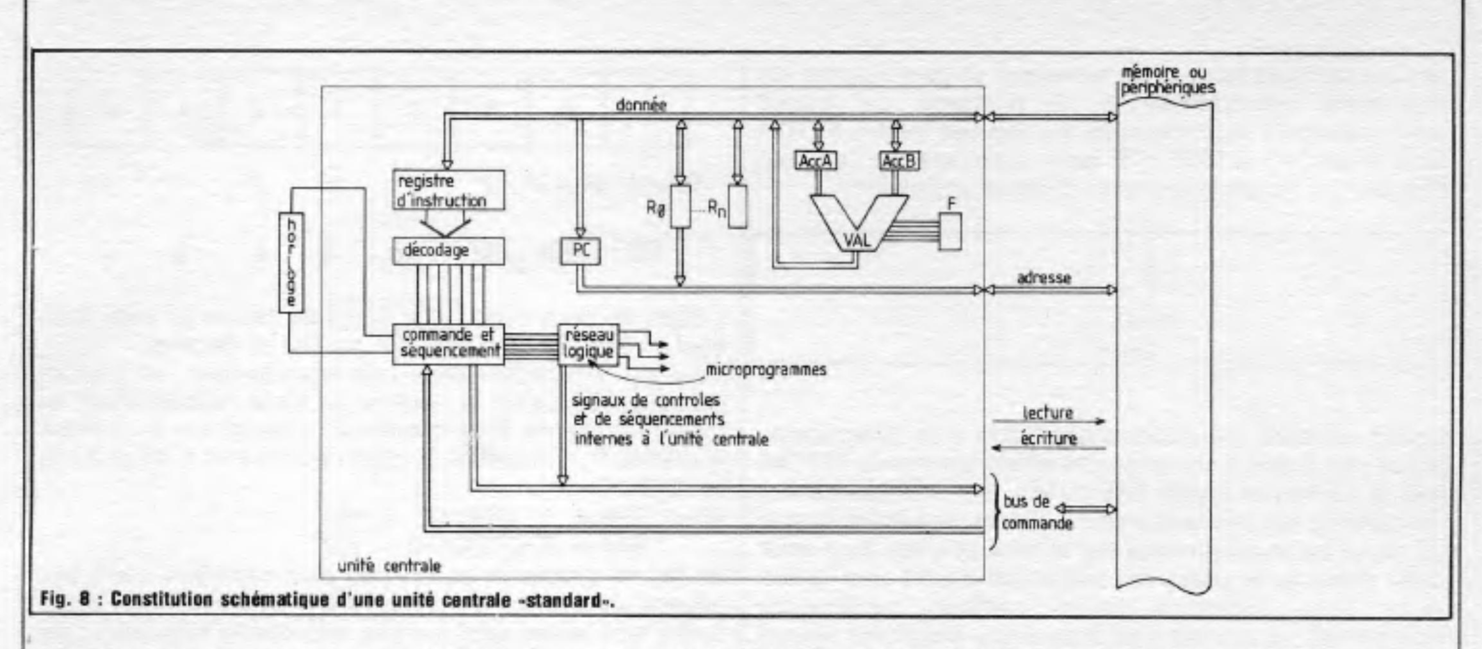


Fig. 8 : Constitution schématique d'une unité centrale «standard».

– ou une ligne de programme = (Charge dans R0) + (3) = une instruction
– ou une ligne de programme = (Stocke à l'adresse 2000) + (la valeur 500) = une instruction
– etc...

Prenons un exemple :

Il s'agit d'additionner 2 avec 2 et de placer le résultat dans la zone données de la mémoire à l'adresse 2000 ; le programme situé à l'adresse 1000 s'écrira ainsi :

Adresse 1000 additionne 2 avec 2
Adresse 1000 + X Stocke le résultat en 2000

Le séquencement sera le suivant :

① L'unité de séquencement a positionné la ligne de lecture à 0 (un 0 est toujours actif), permettant de lire dans la mémoire (on verra par la suite qu'une unité centrale passe la majorité de son temps à lire !). C'est le compteur de programme PC qui positionne le bus d'adresses à la valeur 1000, permettant ainsi de trouver la 1ère instruction (t1). Le code opératoire transite par le bus de données pour être décodé dans le registre d'instructions (t2).

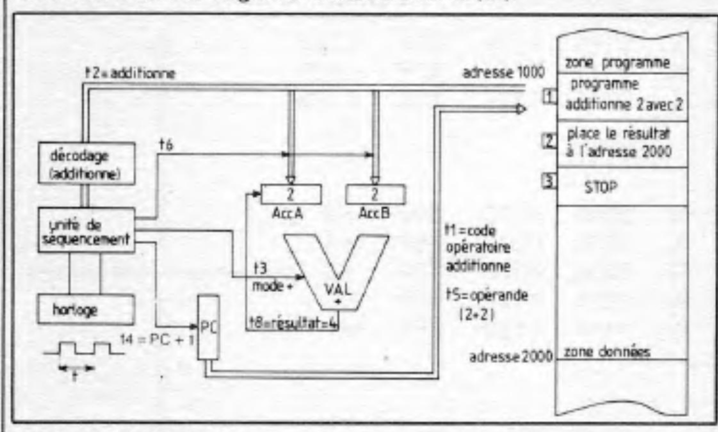


Fig. 9 : Le MOPET additionne.

② L'unité de séquencement positionne l'UAL en mode addition (t3) puis fait avancer le PC d'un pas (t4) pour qu'il puisse aller chercher l'opérande (t5) en mémoire programme, elle ouvre les accumulateurs (t6) de façon qu'ils admettent l'opérande se trouvant enfin sur le bus de données.

Le résultat de l'addition t8 se trouve automatiquement stocké dans l'un des accumulateurs (supposons par convention que ce soit A).

Nous remarquons que l'instruction d'addition s'exécute en 8 séquences. Si une horloge régit chaque séquence, on en déduit que cette instruction s'est exécutée en 8 périodes.

③ L'unité de séquencement a positionné le PC à l'adresse suivante, qui correspond à l'instruction suivante (t1), le code opératoire transite par le bus de données pour être décodé (t2). Puisqu'il s'agit d'une instruction de stockage à une adresse définie (ici 2000), l'unité de séquencement positionne AccA en sortie et aiguille le bus de données sur un registre compteur de programme temporaire (non adressable par le programmeur) (t3).

④ Le PC s'incrémente d'un pas pour prendre l'opérande qui est chargé immédiatement dans le PC temporaire (t4).

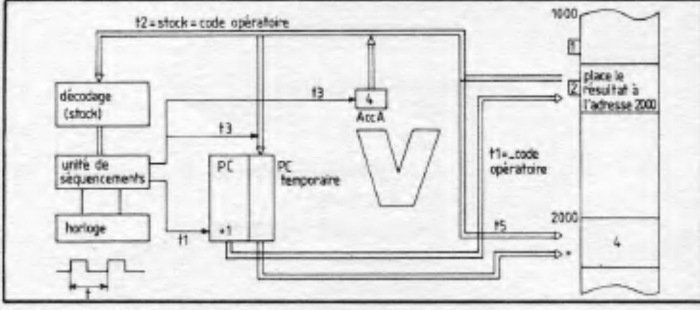


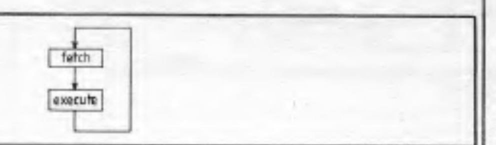
Fig. 10 : Le MOPET en stockage.

La donnée transite ensuite par le bus de donnée pour être stockée à l'adresse pointée par le PC temporaire (t5).

Le travail est fini en 5 périodes d'horloge.

Il a fallu en tout 13 périodes d'horloge pour exécuter une modeste opération d'addition... un meilleur concepteur aurait sûrement fait beaucoup mieux.

Si vous ne l'avez pas encore remarqué, on peut résumer les opérations précédentes en 2 moments ou cycles continuellement recommencés, il s'agit des cycles FETCH (pour chercher) et EXECUTE (pour exécuter) que l'on peut résumer par l'organigramme ci-dessous :



L'unité centrale, en cours d'exécution d'un programme, passe son temps à chercher une instruction (cycle FETCH) puis à l'exécuter (cycle EXECUTE). Son intelligence (en l'occurance son microprogramme interne) ne lui permet pas de savoir qu'un programme est terminé et c'est donc pour cette raison qu'on place une instruction d'arrêt pour lui dire «c'est fini», autrement l'unité centrale continuerait indéfiniment l'exécution d'un programme imaginaire faisant suite aux instructions exécutables. On dira alors que le système est «planté» (pour utiliser le jargon cher aux informaticiens !).

Nous allons maintenant élaborer la partie matérielle de notre MOPET, commençons donc par construire l'Unité Arithmétique et Logique (UAL).

L'UAL du MOPET

Comme son nom l'indique, cette unité doit être capable d'effectuer des opérations Arithmétiques et Logiques ; les plus simples sont : (tableau 1) :

Opérations arithmétiques :
- Multiplication par 2
- Division par 2
- Complémentation à 2
- Addition
- Soustraction
- Comparaisons

Opérations logiques :
- ET
- OU
- Complémentation (à 1)
- Ou Exclusif
- Décalages logiques
- Rotations
- Test de bit

Ces opérations s'effectueront sur un mot de 8 bits appelé octet, contenu dans l'un des accumulateurs.
Chaque bit de cet octet est affecté d'un poids, par exemple le bit numéro 7 est affecté du poids 2^7 = 128, le bit numéro 0 est affecté du poids 2^0 = 1 etc...

7	6	5	4	3	2	1	0
B7 = 2^7	b6 = 2^6	2^5	Z^4	$\cdot 2^3$	2^2	2^1	2^0
128	64	32	16	8	4	2	1

L'octet 85 sera codé 1000 0101 en binaire et sera donc égal à : $2^7 + 2^2 + 2^0 = 128 + 4 + 1 = 133$ en décimal.
Pour s'y reconnaître dans ces équivalences, on prendra l'habitude de coder le nombre en base hexadécimale en utilisant le préfixe $; le nombre en base binaire en utilisant le préfixe % ; le nombre en base décimale en n'utilisant pas de préfixe.

Ainsi : $85 = % 10000101 = 133
$66 = % 01100110 = 102

En fait, le problème est un peu plus complexe car il faut pouvoir représenter les nombres en valeurs signées, c'est-à-dire qu'il faudra tenir compte des valeurs négatives ; par convention, on définira le bit n° 7 comme un bit de signe, ainsi si b7 = 0 le nombre codé sur 8 bits est POSITIF : si b7 = 1, il est NEGATIF.
Notre nombre N pourra donc être compris entre deux extrêmes qui sont :

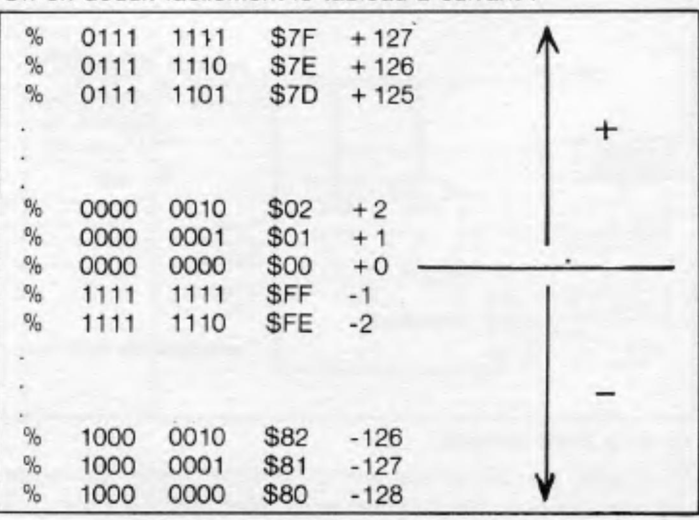
% 01111111 = $7F = $2^0+2^1+2^2+2^3+2^4+2^5+2^6=2^7$

-1 = 127 et % 10000000 = $80 = $-128 = -2^7$

On trouvera facilement ce nombre négatif en complémentant à 1 le nombre positif et en ajoutant 1 au résultat ; ainsi, si N = une valeur positive, N en complément à 2 sera égal à N2's = $\overline{N}$ + 1.

Ex. si N = $00 N2's = $FF + $01 = $00 (ce qui est normal)
si N = $01 N2's = $FE + $01 = $FF (1 et -1 en décimal)
si N = $10 N2's = $EF + $01 = $11 (16 et -16 en décimal)

On en déduit facilement le tableau 2 suivant :



%	0111	1111	$7F	+ 127	↑
%	0111	1110	$7E	+ 126	
%	0111	1101	$7D	+ 125	+
.					
.					
.					
%	0000	0010	$02	+ 2	
%	0000	0001	$01	+ 1	
%	0000	0000	$00	+ 0	
%	1111	1111	$FF	-1	
%	1111	1110	$FE	-2	
.					
.					−
.					
%	1000	0010	$82	-126	
%	1000	0001	$81	-127	
%	1000	0000	$80	-128	↓

Notre MOPET travaillera donc en permanence en complément à 2, il tiendra compte de ce fait dans toutes les opérations arithmétiques qu'on lui demandera d'effectuer et l'ignorera lors d'opérations logiques.
Fort de cet enseignement, nous sommes dorénavant en mesure d'élaborer une Unité Arithmétique et Logique !

Fonctions logiques

Compte tenu du tableau 1, nous devons réaliser des opérations logiques booléennes classiques. L'élément de base sera le transistor unipolaire puisque la technologie utilisée sera du type MOS (figures 11 et 12).

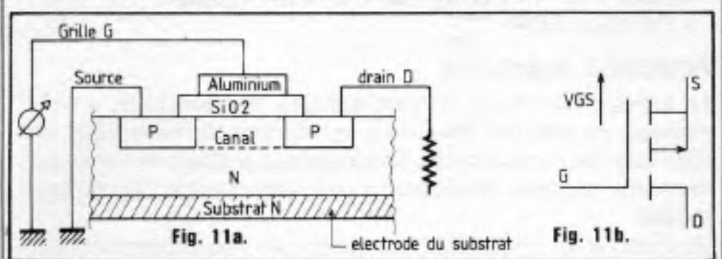


Fig. 11 : Transistor du type PMOS.

Sur un substrat de type N on développe deux ilôts de type P, l'un appelé Source et l'autre Drain. La grille est séparée du substrat par une couche d'isolant qui est de l'oxyde de silicium.
Une tension négative de grille provoque un afflux de porteurs positifs (trous) dans l'espace source-drain (fig. 11a).
Puisque le canal ainsi constitué est riche en trous, on dira que le transistor est du type P, d'où l'appelation PMOS (fig. 11b).

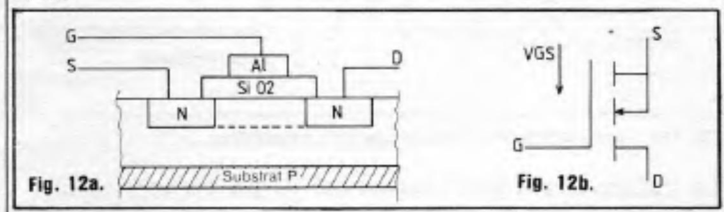


Fig. 12 : Transistor du type NMOS.

En développant deux ilôts de type P sur un substrat de type N, on réalise un transistor NMOS (fig. 12a).
Une tension positive de grille provoquera le passage d'électrons majoritaires dans l'espace source-drain. Ces deux porteurs étant négatifs, le canal formé sera donc du type N d'où l'appellation NMOS (fig. 12b).
Il est aisé de réaliser un inverseur en disposant deux transistors PMOS et NMOS tête bêche (fig. 13).
Ainsi, si E = 0, la tension VGS du NMOS est égale à 0, il est bloqué et offre donc une résistance infinie entre drain et source ; par contre, le PMOS se sature puisque sa tension de grille est plus faible que sa tension de source, on trouve ainsi la tension Vdd sur la sortie S.
Le NMOS va se saturer si E = 1, d'où S = 0.

On réalisera une fonction NAND sans problème en disposant 2 NMOS en cascade : pour vérifier la fonction, il faudra bien que le premier et le 2ᵉ transistor conduisent.

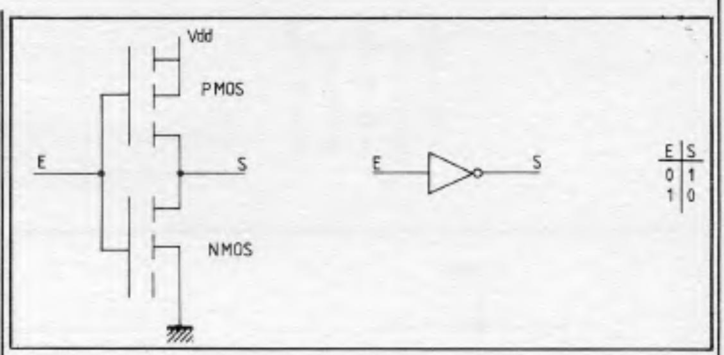
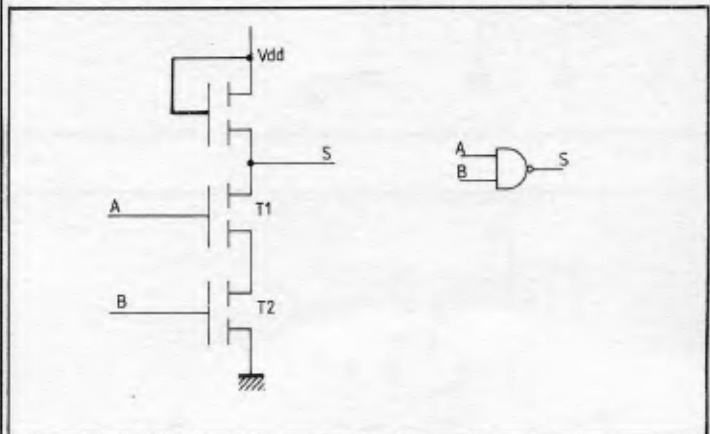


Fig. 13 : Un inverseur.

Remarquons la présence d'une résistance de polarisation qui n'est autre qu'un transistor MOS dont la grille est reliée à la source (fig. 14).



A	B	Sortie T1	Sortie T2	S
0	1	Bloquée	Saturée	1 = Vdd
1	0	Saturée	Bloquée	1 = Vdd
0	0	Bloquée	Bloquée	1
1	1	Saturée	Saturée	0

Fig. 14 : Un NAND.

Un NOR sera réalisé par deux transistors disposés en parallèle (fig. 15). Les fonctions ET, OU seront simples à réaliser à partir des éléments déjà connus (fig. 16a et b).

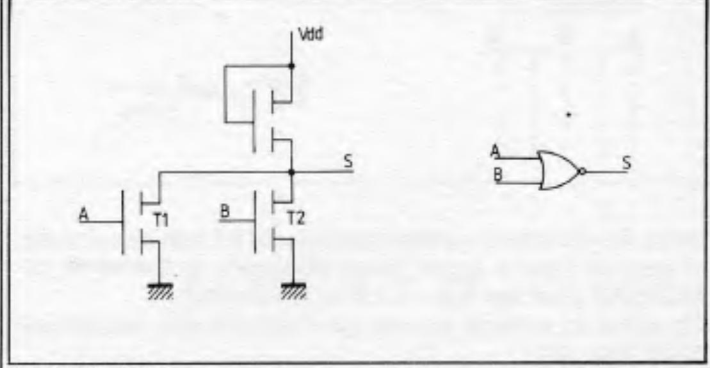
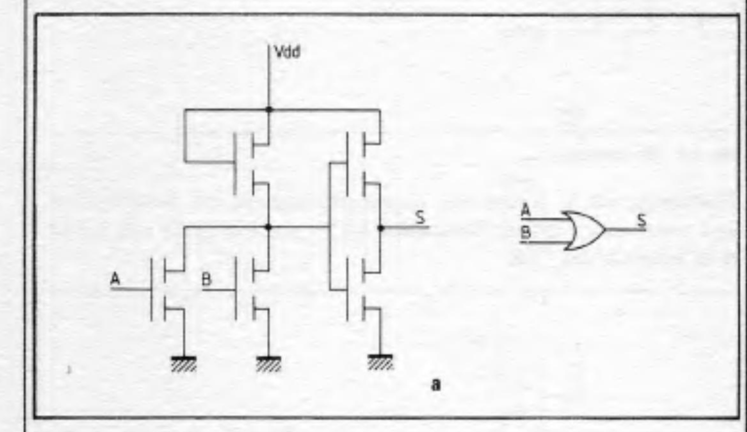


Fig. 15 : Un NOR.

A	B	S
0	1	0
1	0	0
0	0	1
1	1	0



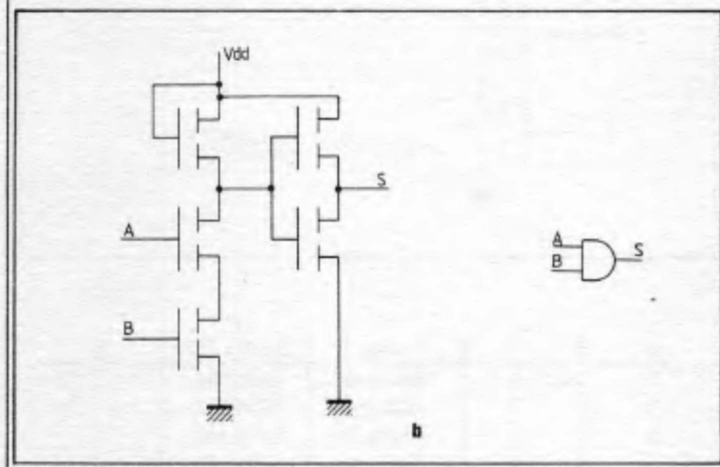


Fig. 16 : (a) un OR (b) un AND.

La dernière fonction combinatoire classique est la fonction OU exclusive (Exclusive OR = EOR), elle est déduite de la table de vérité suivante :

Entrées

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

A, B : $S = A\bar{B} + \bar{A}B = A \oplus B$

Cette fonction sera réalisée simplement à l'aide des circuits ci-dessus. Pour y arriver, nous utiliserons le théorème de MORGAN pour qui $\overline{A.B} = \bar{A} + \bar{B}$ et $\overline{A + B} = \bar{A}.\bar{B}$

On arrive au schéma suivant qui n'utilisera que des portes NAND (fig. 17) :

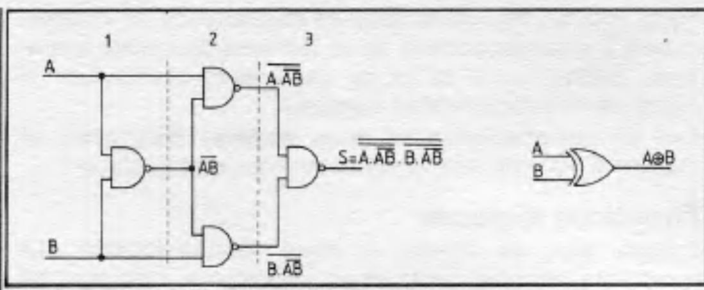


Fig. 17.

$$S = \overline{\overline{A.\overline{AB}}\ .\ \overline{B.\overline{AB}}} = \overline{\overline{A.\overline{AB}}} + \overline{\overline{B.\overline{AB}}} = A.\overline{AB} + B.\overline{AB}$$
$$= \overline{AB}(A + B) = (\bar{A} + \bar{B})\ (A + B) = \bar{A}B + \bar{B}A = A \oplus B$$
= Fonction EOR

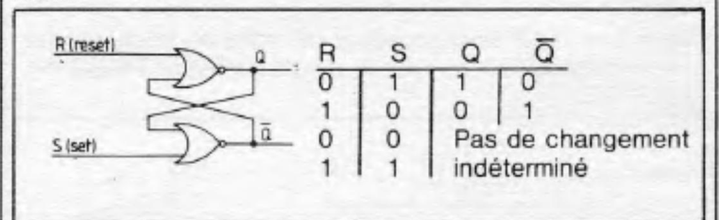
Fonction mémoire

La logique décrite ci-dessus est dite combinatoire, il nous manque un élément important qui permet de mémoriser les résultats des opérations combinatoires, il s'agit de l'élément mémoire obtenu simplement en introduisant le facteur temps.

En effet, si en un temps t, on mémorise une information que l'on pourra récupérer à l'instant t + 1 sans aucune modification, on aura certes perdu l'aspect fugitif de cette information, mais on aura gagné une nouvelle possibilité qui permet de mémoriser indéfiniment une information.

Un tel élément capable de stocker une information binaire s'appelle une mémoire ou bascule ou flip flop ou latch.

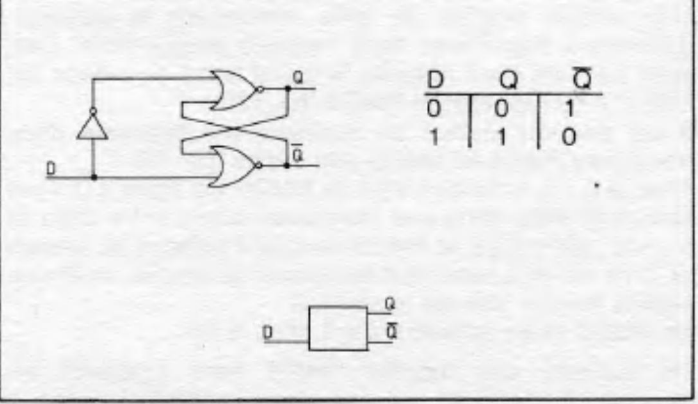
La figure 18a représente une bascule avec des portes NOR.



R	S	Q	$\bar{Q}$
0	1	1	0
1	0	0	1
0	0	Pas de changement	
1	1	indéterminé	

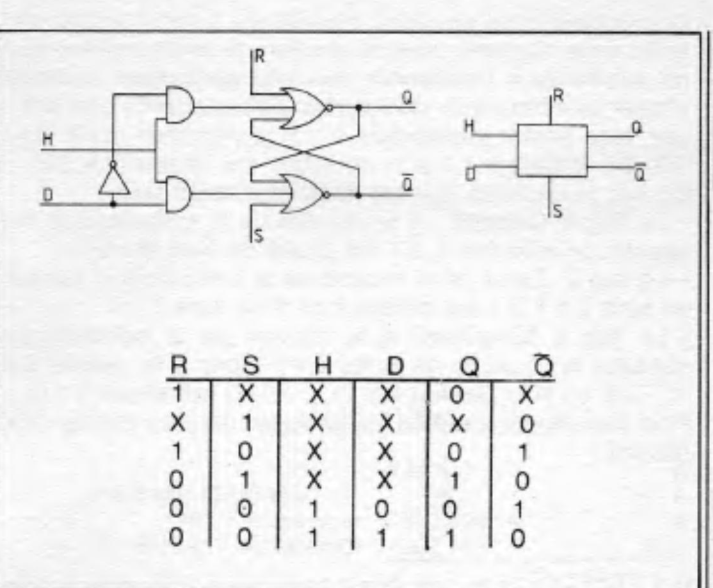
Fig. 18a : Une bascule RS n'accepte que trois possibilités.

Le malheur d'une telle bascule est qu'elle accepte un état indéterminé (R = S = 1) que l'on pourra supprimer en obligeant R et S à avoir deux états complémentaires, il s'agit de la bascule D (fig. 18b).



D	Q	$\bar{Q}$
0	0	1
1	1	0

Fig. 18b : Bascule type D.



R	S	H	D	Q	$\overline{Q}$
1	X	X	X	0	X
X	1	X	X	X	0
1	0	X	X	0	1
0	1	X	X	1	0
0	0	1	0	0	1
0	0	1	1	1	0

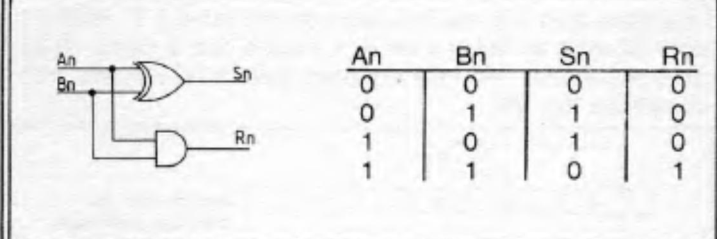
Fig. 18c : Bascule D synchronisée sur une horloge avec SET et RESET.

Pour rester synchrone sur des évènements extérieurs, il suffira d'ajouter une entrée horloge, de même qu'une entrée Set (S) initialisera la sortie Q de la bascule à 1 et Reset (R) à 0.
C'est une bascule de ce type qui constituera les registres et accumulateurs de notre MOPET (fig. 18c).

Fonction arithmétique

L'addition

Une première opération à réaliser est l'addition ; en raisonnant sur les bits de rang n de 2 valeurs A et B, on obtient le tableau de la figure 19.

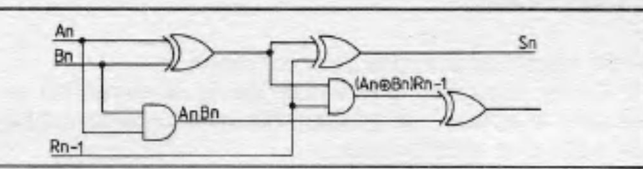


An	Bn	Sn	Rn
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Fig. 19 : Un demi-additionneur.

Rn étant la retenue, on obtient deux équations :
$Sn = An\ \overline{Bn} = \overline{An}\ Bn = An \oplus Bn$
$Rn = An\ Bn$

L'additionneur étudié est encore incomplet puisqu'il faut tout de même tenir compte de la retenue provenant de l'addition sur les bits de poids n-1.
On arrive ainsi au tableau de la figure 20.



An	Bn	Rn-1	Sn	Carry Rn
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$Sn = \overline{An}\ \overline{Bn}\ Rn\text{-}1 + \overline{An}\ Bn\ \overline{Rn\text{-}1} + An\ \overline{Bn}\ \overline{Rn\text{-}1} + An\ Bn\ Rn\text{-}1$
$Rn = An\ Bn\ Rn\text{-}1 + An\ \overline{Bn}\ Rn\text{-}1 + An\ Bn\ \overline{Rn\text{-}1} + An\ Bn\ Rn\text{-}1$
$Sn = Rn\text{-}1\ (\overline{An}\ Bn + An\ Bn) + Rn\text{-}1\ (\overline{An}\ Bn + \overline{Bn}\ An)$
$S = Rn\text{-}1\ (\overline{An \oplus Bn}) + \overline{R}n\text{-}1\ (An \oplus Bn)$
$S = Rn\text{-}1 \oplus (An \oplus Bn)$
$Rn = An\ Bn\ (\overline{Rn\text{-}1} + Rn\text{-}1) + Rn\text{-}1\ (\overline{An}\ Bn + An\ \overline{Bn})$
$Rn = An\ Bn + Rn\text{-}1\ (An \oplus Bn)$ **Fig. 20 : Additionneur binaire de poids N.**

Un additionneur complet sera donc formé d'une chaîne d'étages additionneurs identiques à celui de la figure 20.
Sur la figure 21, on voit que la ligne de départ à 1 donnera les informations A1, B1, A2, B2... An, Bn simultanément aux additionneurs.
Compte tenu du délai d'addition, la ligne résultat qui passera à 1 permettra d'enregistrer la somme S0, S1, S2... Sn dans une mémoire.
On se réservera aussi la possibilité d'assurer l'addition en tenant compte d'une carry éventuelle (appelée ici Cy) provenant d'une addition précédente (fig. 21).
En effet, si nous faisons :

$80		1000	0000
+ $80	+	1000	0000
[1] $00 = R	[1]	0000	0000 = R
C	C		

Nous obtenons en réalité la valeur $100 qui sera codée sur 9 bits ! Comme cela n'est pas possible, nous nous contenterons d'un résultat sur 8 bits en mémorisant toutefois la carry qui nous sera bien utile si nous codons le résultat sur 16 bits.

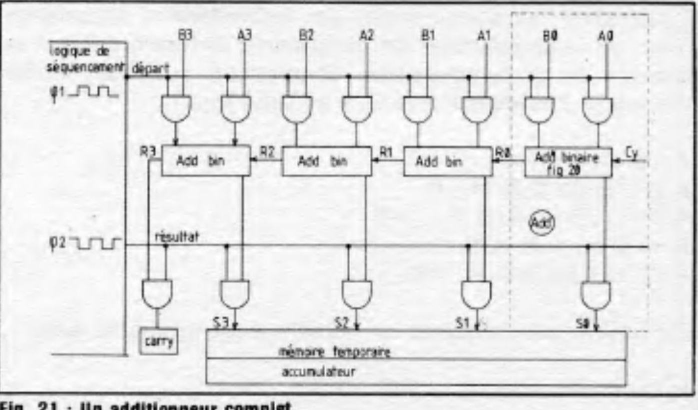


Fig. 21 : Un additionneur complet.

La soustraction

A ce stade, nous avons deux solutions :

1) Ecrire l'équation booléenne de la soustraction et en déduire le schéma de principe qui sera forcément différent de celui d'un additionneur.

2) Garder la structure de l'additionneur en ajoutant l'élément soustraction. C'est bien entendu la seconde solution qui sera choisie puisqu'elle part d'un principe fort simple : pour soustraire, on va additionner !

Nous avons vu qu'un nombre peut être représenté en valeurs négatives en utilisant le complément à 2 ; la plage de variation d'une valeur binaire codée sur 8 bits sera comprise entre + 127 et -128 d'où $A-B = A + (\overline{B} + 1)$, on conserve ainsi l'additionneur binaire comme le montre clairement la figure 22.

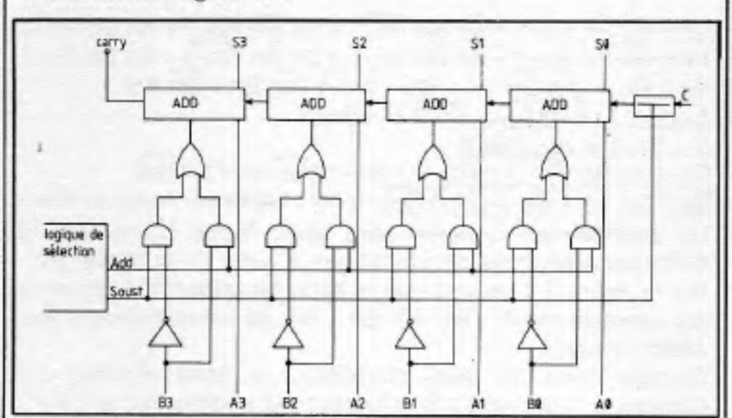


Fig. 22 : L'additionneur-soustracteur.

Si Add = 1 → Soust = 0 on obtient Si = Ai + Bi (la carry C éventuelle est ajoutée sur l'élément ADD de poids le plus faible).

Si Add = 0 → Soust = 1 on obtient $Si = Ai + (\overline{Bi} + 1)$ la valeur 1 est ajoutée au premier élément ADD.

Chaque élément ADD représente l'intérieur de l'espace entre pointillés de la figure 21.

La possibilité Add = Soust = 1 est impossible à exécuter grâce à la logique de sélection, par contre Add = Soust = 0 inhibe l'additionneur soustracteur.

La comparaison

L'opération de comparaison est extrêmement importante pour qui veut exécuter un programme différent suivant le résultat de la comparaison. Supposons que l'on veuille comparer 2 nombres A et B. Il se peut que :

A = B ou A-B = 0
A > B ou A-B > 0
A > = B ou A-B > = 0
A < B ou A-B < 0
A < = B ou A-B < = 0

On peut aussi comparer un nombre avec lui-même ainsi :

A = 0;A > 0;A > = 0;A < 0;A < = 0

Une comparaison est donc équivalente à une soustraction entre deux nombres, mais le résultat de cette soustraction ne servira qu'à positionner des bits particuliers nommés «flags» ou «drapeaux» qu'il suffira de tester si besoin est. Les flags seront positionnés à 1 si la condition reste vraie (TRUE), ils seront à 0 si la condition est fausse (FALSE). On voit ci-dessous que les flags importants sont :

– Le flag N (négatif) : si le résultat de la comparaison est négatif, on aura N = 1, s'il est positif on aura N = 0

– Le flag Z (Zero) : si le résultat de la comparaison est nul, on aura Z = 1, s'il est différent de 0 on aura Z = 0

– Le flag V (Overflow), si le résultat de la comparaison dépasse la capacité de 8 bits V = 1 (donc si le résultat est < -128 ou si le résultat est > + 127) autrement V = 0

Pour bien comprendre ce qui se passe dans ce dernier cas, faisons :

A	= – 123 et B	= + 15
A – B	= – 138	< – 128 (limite des 8 bits)
A	= 1000 0101	représente – 123
– B	= 1111 0001	représente – 15 ($-B = \overline{B} + 1$)

0 1 1 0111 0110 = une valeur positive (+ 118) alors qu'elle devrait être négative.

V sera donc le ou exclusif de N et C → $V = N \oplus C$ (entraînez-vous avec d'autres exemples).

Nous avons tous les éléments en main pour réaliser l'opération de comparaison. Construisons le tableau ci-dessous :

A	B	A = B A – B = 0	A > B A – B > 0	A ⩾ B A – B ⩾ 0	A < B A – B < 0	A ⩽ B A – B ⩽ 0	N	Z
0	0	1	0	1	0	1	0	1
0	1	0	0	0	1	1	1	0
1	0	0	1	1	0	0	0	0
1	1	1	0	1	0	1	0	1

Pour A = B on a $\overline{A}\,\overline{B} + AB = \overline{A \oplus B} = S$

Pour A > B on a $A\overline{B} = S1$

Pour A < B on a $\overline{A}B = S2$

Pour A ⩽ B on a $\overline{A}\,\overline{B} + AB + \overline{A}B = S + S2$

Pour A ⩾ B on a $\overline{A}\,\overline{B} + AB + A\overline{B} = S + S1$

La condition Z = 1 est donnée par S qui représente l'équation d'un 1/2 additionneur complémenté à 1... et N = 1 pour S2 mais en fait N n'est rien d'autre que la copie du bit de poids le plus fort ! Ce qui nous évitera un schéma trop compliqué (fig. 23).

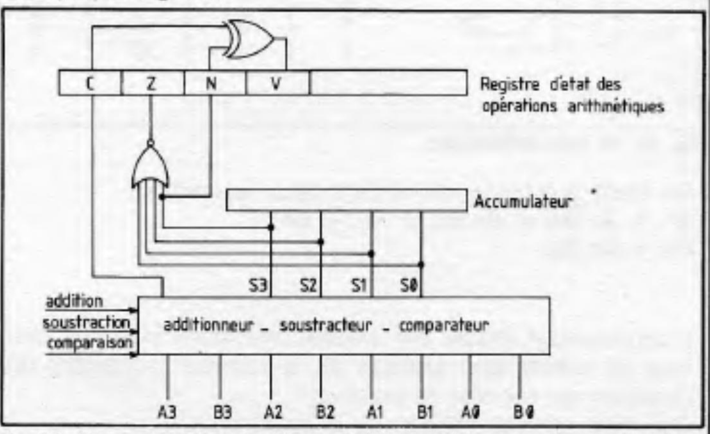


Fig. 23 : Une comparaison des 2 mots linéaires positionne les flags Z, N et V.

Les décalages

Pour ne pas compliquer le problème outre mesure, il serait possible de réaliser un multiplicateur et un diviseur 8 bits par 8 bits avec le résultat sur 16 bits dans le 1er cas, et sur 8 bits dans le second cas + 8 bits de reste.
Mais il est cependant plus simple d'aboutir à un résultat identique à une multiplication et à une division en effectuant de simples décalages. En effet, chaque élément binaire d'un nombre codé sur 8 bits représente un poids qui est un multiple de 2.
En lisant les bits de la droite vers la gauche, on trouve :
bit 0 = b0 = 2^0 = 1
bit 1 = b1 = 2^1 = 2
bit 2 = b2 = $2^2 = 2^1 \times 2^1$
bit 3 = b3 = $2^3 = 2^1 \times 2^1 \times 2^1$ etc...
Donc, chaque déplacement sur la gauche représente une puissance de 2 (ce qui fait qu'on multiplie chaque fois par 2) de même, chaque déplacement sur la droite représente une division par 2.
Par exemple : 12 en décimal donne

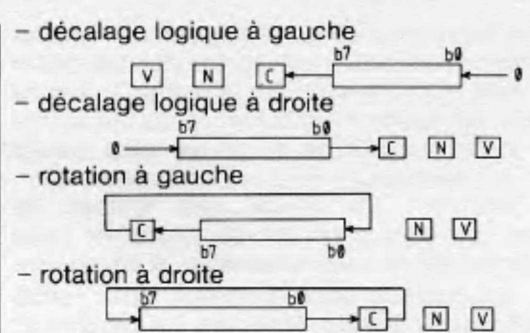
0	0	0	0	1	1	0	0
128	64	32	16	8	4	2	1

En faisant un décalage à gauche, on multiplie bien par 2 puisqu'on obtient :

C V N   0 0 0 0 0 01 1 0 0 0
0 0 0

ce qui donne bien 24 en décimal.
On remarquera qu'un tel décalage impose un zéro introduit sur le bit de poids le plus faible alors que le bit de poids fort est introduit dans la carry, puisque b7 = 0, on obtient N = 0 d'où V = N ⊕ C = 0.
Si on prend maintenant un nombre négatif (-12 par exemple) qui s'écrira : C ← 1 1 1 1 0 1 0 0 ← 0, un décalage à gauche donnera

V N C
0 1 1 ← 1 1 1 0 1 0 0 0 ← qui donne bien -24...
si V = 1 cela voudrait dire qu'il y a dépassement de capacité (résultat < -128).
On pourrait penser qu'un décalage à droite décalqué sur le même principe que le décalage à gauche pourrait suffire pour la division par 2 ; mais c'est oublier la présence du bit de signe qui impose de le dupliquer après chaque décalage afin d'obtenir un résultat correct.
Ainsi +12 donne 0 0 0 0 1 1 0 0 après un décalage à droite, on obtient :
0 0 0 0 0 1 1 0 → 0 0 ce qui donne bien 6 (V
C N
n'aura pas lieu dêtre positionné pour une division !).
Mais -12 donne 1 1 1 1 0 1 0 0 après un décalage à droite on obtient :
1 1 1 1 1 0 1 0 → 0 1 ce qui donne bien -6.
C N
Ces deux fonctions de décalage sont du type arithmétique. On peut aisément introduire des fonctions logiques en les définissant ainsi :



On remarquera que le décalage logique à gauche est identique au décalage arithmétique à gauche, mais par contre le décalage arithmétique à droite est différent du décalage logique qui introduit un zéro sur le bit de poids fort (c'est donc pour cette raison que l'on aura toujours N = 0).
Le but d'un décalage logique est de faire «tomber» un bit dans la carry, qu'il suffira ensuite de tester pour voir si ce bit est à 1 ou 0 (il existera bien sûr une instruction spécifique testant spécialement cette carry).
Un décalage logique change le mot contenu dans le registre, par contre, la rotation permet de récupérer ce mot après 8 tours.
Nous avons maintenant assez d'éléments pour construire le dernier élément de notre unité arithmétique et logique.

Le registre (appelé encore accumulateur)

On appelle registre une mémoire linéaire à accès parallèle. Chaque case du registre est constituée d'une bascule accessible séparément, mais il est aussi possible de déplacer pas-à-pas l'information dans ce registre, de même qu'il est facile de l'utiliser en entrée-sorties parallèles.
La figure 24 donne un exemple de registre à chargement parallèle et décalage à droite.

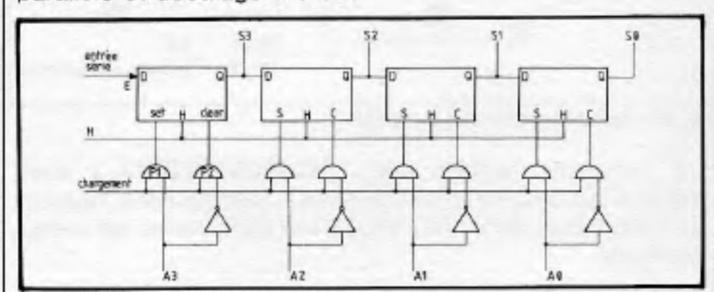


Fig. 24 : Un registre parallèle-série.

· Chargement parallèle : un ordre de chargement permet de recopier le mot présent en A3A2A1A0 sur S3S2S1S0.
si A3 = 1, la porte P1 est ouverte, d'où set = 1 et Q = 1
si A3 = 0, la porte P2 est ouverte, d'où clear = 1 et Q = 0.
Dans ces 2 cas, on doit avoir la condition chargement = 1.
Décalage à droite : une impulsion sur la borne H des bascules type D permet de copier l'état présent sur l'entrée E vers l'entrée suivante F. Donc, 4 impulsions d'horloge auront transporté l'information E sur la sortie S0.
Une électronique supplémentaire permettrait d'assurer un décalage à gauche et une rotation dans les deux sens mais notre but n'est pas de trop compliquer un problème qui l'est déjà assez !
Une toute dernière remarque qui a son importance : les bascules en technologie MOS ne peuvent conserver

indéfiniment une information sans rafraîchissement interne. En effet, une information binaire est représentée par un «1» logique sur la grille d'un transistor, or, la capacité interne d'un tel transistor est relativement faible, ce qui fait qu'elle se décharge assez rapidement et on va ainsi perdre définitivement l'information que nous voulions mémoriser.

Il est donc important de prévoir une horloge de rafraîchissement qui aura pour but de recharger cette capacité afin de conserver cette information si importante.

Les registres, qui peuvent être nombreux dans l'unité centrale, doivent pouvoir être déconnectés, il s'agit d'un 3e état qui introduit un nouveau signal de contrôle appelé «three state control» (TSC).

Cette fonction est très intéressante lorsque l'on branche plusieurs registres en parallèle : il suffit de placer en 3e état tous les registres non utilisés.

La figure 25 donne un exemple d'une porte à 3 états : la fonction 3 états est réalisée par une porte de transfert CMOS branchée en série avec la sortie.

Suivant la valeur de TSC, les deux transistors sont conducteurs et on a $S = \overline{AB}$... ou les deux transistors sont bloqués et la porte NAND est en 3e état.

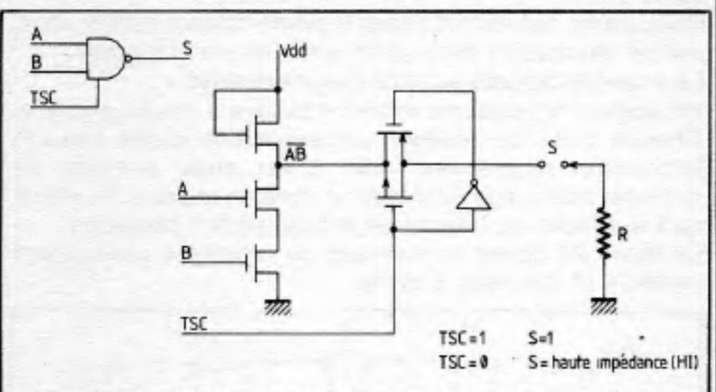


Fig. 25 : Une porte NAND avec borne TSC.

De nombreux buffers (type 74LS640-74LS645...) sont réalisés sur ce principe et on trouve, comme nous l'avions dit, l'équivalent dans notre MOPET et dans toutes les unités centrales.

Réalisation d'une ALU complète

Une ALU complète simplifiée (celle de notre MOPET) est présentée figure 26a, par contre la figure 26b montre le schéma block d'une ALU classique à deux entrées : les buffers sont des registres temporaires normalement inaccessibles par le programmeur et le fonctionnement de cette ALU est présenté avec un accumulateur. On pourrait prévoir un deuxième accumulateur présent sur l'entrée B qui offrirait la possibilité de travailler avec des mots de 16 bits au lieu de 8.

L'unité de commande du MOPET

Nous avons vu précédemment que tout ordre envoyé à l'unité centrale se traduit par deux mots binaires qui sont :
- le code opératoire COP
- l'opérande OP

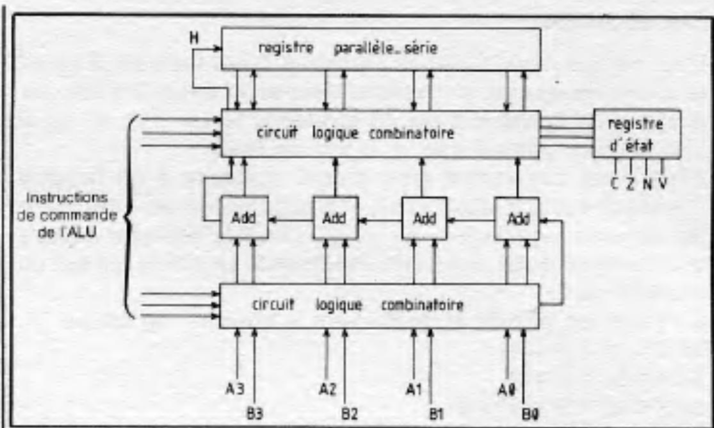


Fig. 26a : Une VAL simplifiée pouvant réaliser toutes les opérations logiques et arithmétiques.

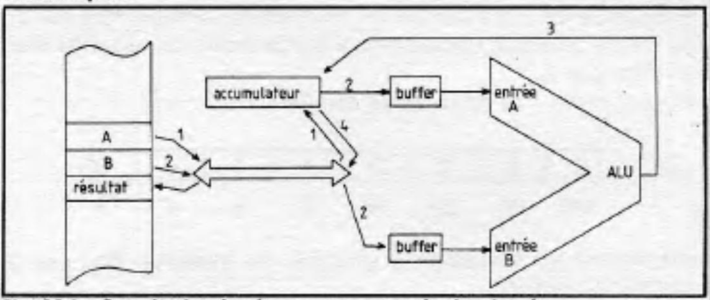


Fig. 26 b : Organisation des étapes pour une opération donnée.
① A→ Accumulateur ③ Résultat à la sortie de l'Alu dans l'accumulateur.
② B→ Buffer→ entrée B alu ④ Résultat Accu→ mémoire.
Accu→ Buffer→ entrée A alu

Fig. 27 : Rôle de l'unité de contrôle.

Nous verrons plus tard que l'opérande n'est, dans certains cas, pas nécessaire, de même que l'association COP + OP peut représenter plus de deux octets.

Le code opératoire représentera le type d'opération que l'UAL devra effectuer, tandis que l'opérande sera la donnée avec laquelle l'UAL devra travailler (rappelons que l'UAL a pour but essentiel, le traitement, la manipulation et la gestion des données : elle peut réaliser des fonctions logiques, arithmétiques, des comparaisons et des tests).

Si ce code opératoire est codé sur 8 bits, on a donc 256 types différents d'opérations à effectuer (de 00 à FF), ce qui

est intéressant mais le sera encore plus si le codage se fait sur 16 bits car on disposera alors de 65536 possibilités différentes !
La figure 27 montre le rôle joué par cette unité de contrôle. Une «loupe» disposée au-dessus de cette unité nous permet d'aboutir au schéma de la figure 8 :

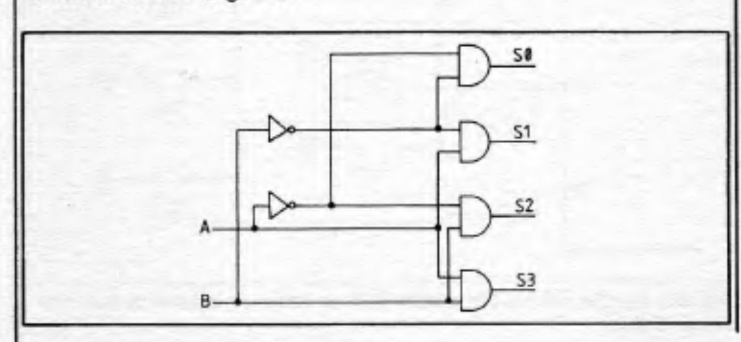
L'unité de contrôle est constituée :
- d'un registre d'instructions RI qui stocke les instructions issues de l'emplacement mémoire adressé par l'unité centrale (son compteur de programme PC). Le RI n'est rien d'autre qu'un latch.
- d'un décodeur nécessaire au décodage du mot contenu dans le RI.
- d'un séquenceur central qui génère des micro-instructions indispensables au bon cheminement des informations. Notons que le séquenceur est du type synchrone, c'est-à-dire que les micro-instructions sont émises au rythme de l'horloge.
En résumé, le séquenceur central est l'organe qui génère les micro-instructions nécessaires au bon cheminement des informations.

Décodage de l'instruction

Nous avons vu que le compteur de programme pointe sur le code opératoire disposé en mémoire.
Le code opératoire représentera une instruction particulière, par exemple un chargement dans l'accumulateur A d'une valeur $3F.
La valeur $3F étant l'opérande, se trouvera à l'adresse suivante c'est-à-dire à PC + 1.
Ce qui peut s'écrire : LDA # $3F en langage assembleur ou 86.3F en hexadécimal.
Laissons tomber pour l'instant le langage assembleur et examinons les valeurs hexadécimales.
Le code opératoire est égal à 86 et l'opérande = 3F.

Décodeur 1 parmi N

La valeur 86 est mise dans le registre d'instructions RI puis est décodée à l'aide d'un décodeur digital à l'entrée duquel on applique le COP de 8 bits. La sortie sera donc constituée de $N = 2^8 = 256$ fils.
Une seule sortie se trouve à 1 pour chacune des valeurs du code d'entrée (fig. 28).



entrées		sorties			
A	B	S0	S1	S2	S3
0	0	1	0	0	0
1	0	0	1	0	0
0	1	0	0	1	0
1	1	0	0	0	1

$S0 = \bar{A}\bar{B}$
$S1 = A\bar{B}$
$S2 = \bar{A}B$
$S3 = AB$

Fig. 28 : Exemple d'un décodeur 1 parmi 4.

Le problème, c'est qu'on a à décoder toutes les valeurs comprises entre 00 et FF d'où une logique plutôt lourde et 256 fils en sortie prennent beaucoup de place sur une puce de silicium de 5 mm × 5 mm !

Décodeur à mémoire morte

On va donc tâcher de diminuer la dimension de ce décodeur en faisant preuve d'astuces.
Par exemple, une matrice de diodes peut représenter un excellent décodeur (fig. 29).

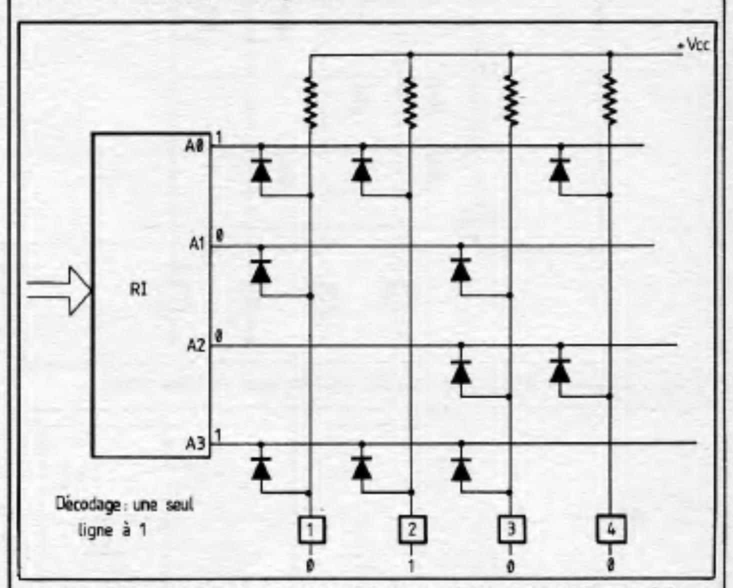


Fig. 29 : Décodeur à diodes : la matrice à diodes décode le mot 1001 placé dans la RI en mettant la ligne 2 à 1 et les autres à 0.

Malheureusement, la ligne 2 restera à 1 tant que A0 et A3 = 1, ce qui rend possible les cas où A1 = 0 et A2 = 0, soit les possibilités suivantes :

1001 ligne 2 = 1
1101 ligne 2 = 1
1111 ligne 2 = 1
1011 ligne 2 = 1

Pour éviter cette incertitude, on décode simultanément le code et son inverse.
Le décodage est dans ce cas parfaitement défini et correspond au code $A3.\overline{A2}.\overline{A1}.A0$ (fig. 30).

Une autre solution consiste à utiliser une matrice de transistors MOS disposés en fonction logique OU (fig. 31). Chaque MOS est commandé par la grille et constitue un point mémoire :
- si la couche d'oxyde est épaisse, le transistor est difficile à saturer
- si elle est mince, le transistor se sature facilement (fig. 32).

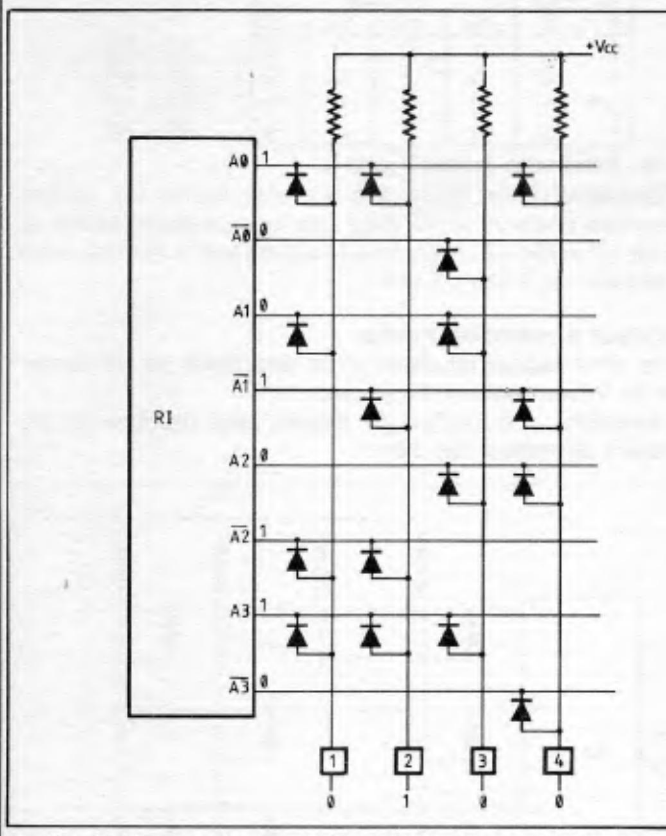


Fig. 30 : Décodeur à diodes : un seul cas possible pour un code donné.

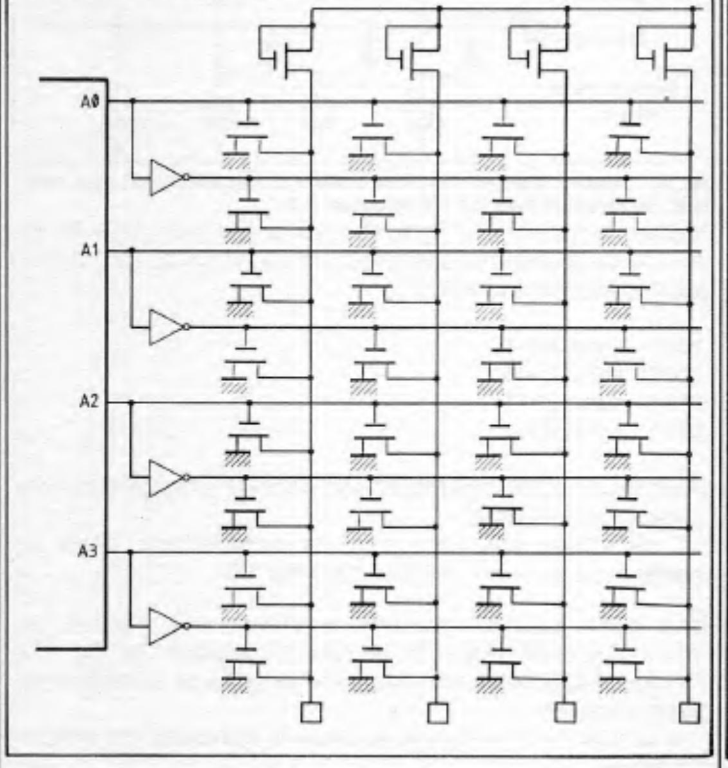


Fig. 31 : Décodeurs à transistors MOS.

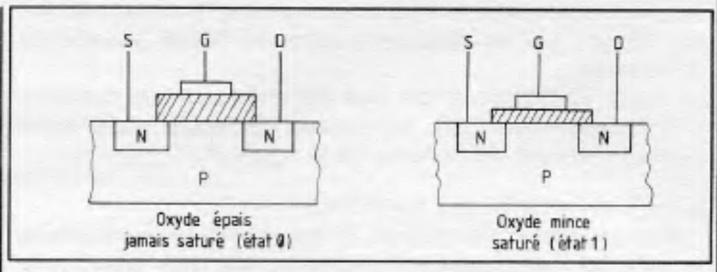


Fig. 32 : Point mémoire d'un décodeur.

Examinons le décodage de LDA (= \$86 = 10000110)

$$LDA = D7\ \overline{D6}\ \overline{D5}\ \overline{D4}\ \overline{D3}\ D2\ D1\ \overline{D0}$$

La matrice de décodage étant constituée par des fonctions OU, nous pouvons transformer cette expression en utilisant le théorème de Morgan de façon à ne faire apparaître que des fonctions OU logiques :

$$\overline{LDA} = \overline{D7.\overline{D6}.\overline{D5}.\overline{D4}.\overline{D3}.D2.D1.\overline{D0}} = \overline{D7} + D6 + D5 + D4 + D3 + \overline{D2} + \overline{D1} + D0$$

$$\overline{LDA} = (\overline{D7} + \overline{D2} + \overline{D1}) + (D6 + D5 + D4 + D3 + D0)$$

Le schéma logique de $\overline{LDA}$ est donné fig. 33 (les points mémoire 0 sont encadrés) :

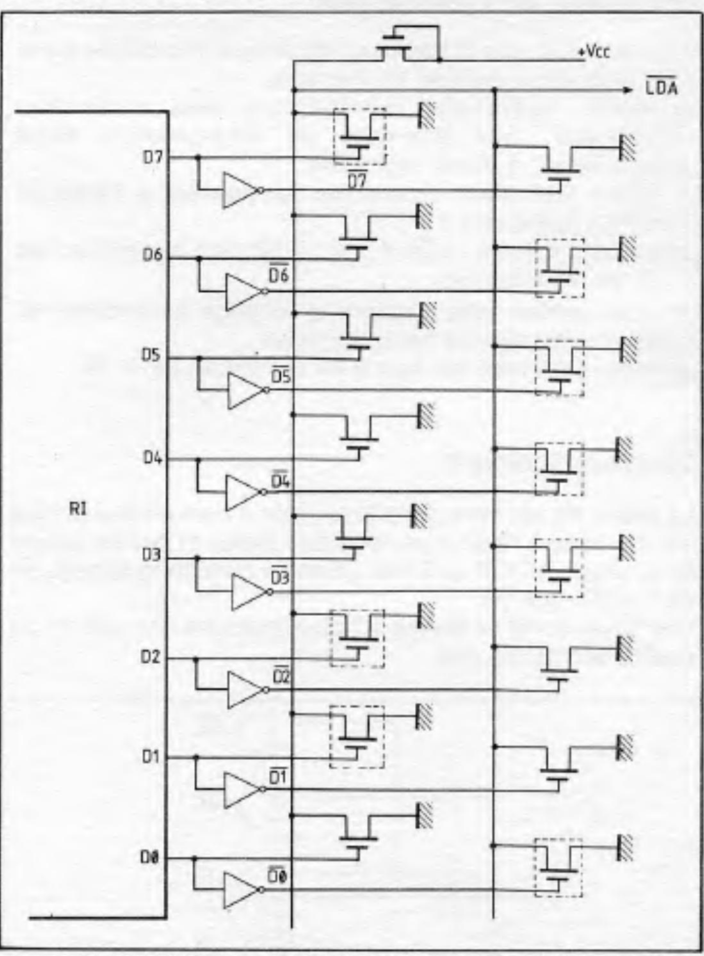


Fig. 33 : Fonction $\overline{LDA}$ sur une matrice MOS (un transistor encadré ne peut être saturé (état 0)).

Décodeur en structure PLA (Programmable Logic Array)

Un réseau logique programmable est basé sur le principe suivant : toute fonction logique peut s'écrire sous forme d'une **somme de produits.**
Si on prend $LDA = D7\ \overline{D6}\ \overline{D5}\ \overline{D4}\ \overline{D3}\ D2\ D1\ \overline{D0}$, on arrive à :
$LDA = (D7.D2.D1).(\overline{D6}.\overline{D5}.\overline{D4}.\overline{D3}.\overline{D0})$
$\overline{LDA} = \overline{(D7.D2.D1).(\overline{D6}.\overline{D5}.\overline{D4}.\overline{D3}.\overline{D0})}$
$\overline{LDA} = (\overline{D7.D2.D1}) + (\overline{\overline{D6}.\overline{D5}.\overline{D4}.\overline{D3}.\overline{D0}})$
Nous avons une relation équivalente à une somme de produits. Il est alors possible de traduire cette équation par un schéma regroupant une matrice produit (ET logique) et une matrice somme (OU logique), fig. 34.

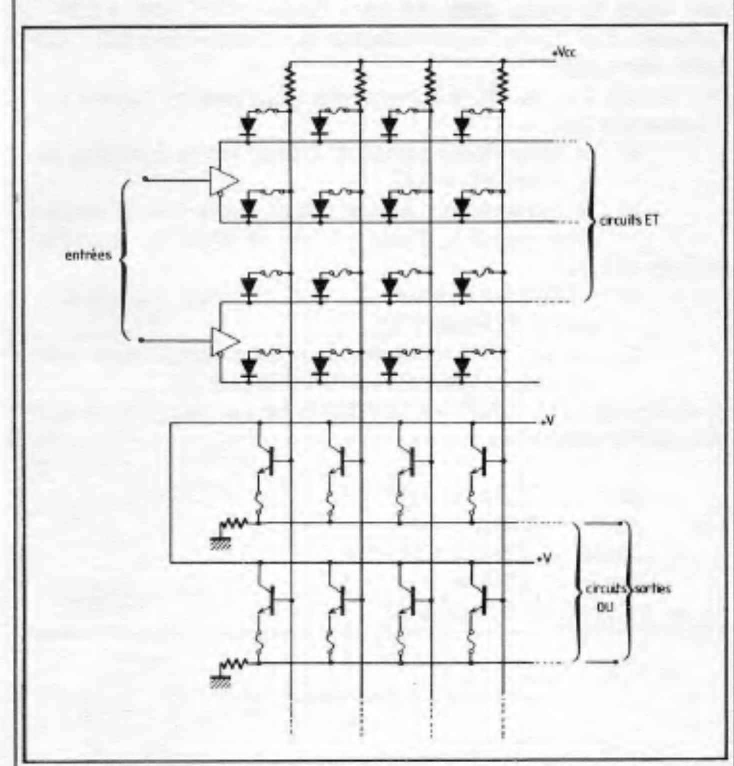


Fig. 34a : Une structure PLA à diodes et transistors à fusibles.

Les portes ET sont réalisées à l'aide d'un réseau à diodes programmables par l'utilisateur selon la technologie à fusibles.
A chaque entrée se trouve un inverseur pour avoir la variable et son complément, ce qui est très utile dans l'écriture des équations logiques.
Les portes OU sont réalisées par des transistors montés en émetteurs suiveurs.
Si on a par exemple un PLA à 8 entrées et 8 sorties, on réalisera $2^8 \times 8 = 2^8 \times 2^3 = 2^{10} \times 2 = 2K$ combinaisons.
Ceci est donc équivalent à une mémoire ROM à 2K points mémoires ou de 2K bits.
Un PLA occupe donc moins de surface qu'une ROM à capacité égale et c'est cette solution qui sera choisie dans l'unité centrale du MOPET, mais au lieu d'utiliser des diodes, il sera très facile d'utiliser des MOS : dans ce cas, la matrice produit est constituée de NAND (fig. 35).

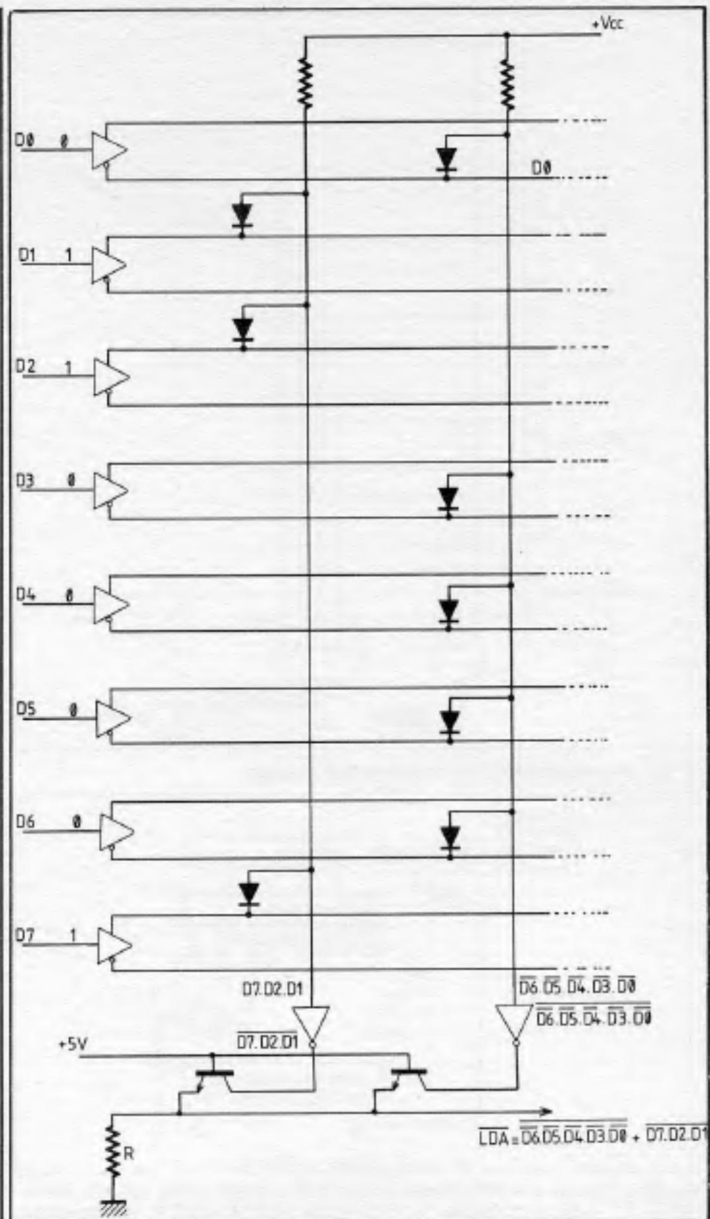


Fig. 34b : Fonction LDA dans une structure PLA à diodes fusibles.

Le séquenceur

Il s'agit d'un élément très important de l'unité centrale. En effet, le séquenceur doit fournir :
– tous les signaux indispensables pour le milieu extérieur tels que :
* un Read/Write (Read = 1 Write = 0) pour lire une mémoire ou écrire.
* un signal périodique à la fréquence de l'horloge interne pour être synchrone avec l'extérieur (nous verrons plus loin que ce problème est un peu plus complexe).
* l'indication de l'état interne de l'unité centrale (fetch, execute, arrêt...)
Il doit aussi mettre l'unité centrale dans un certain état

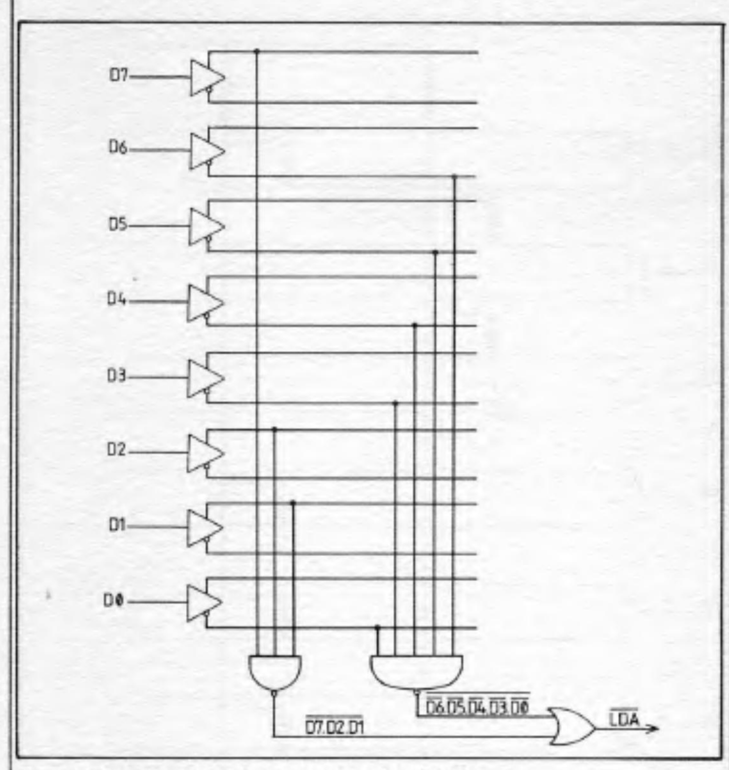


Fig. 35 : Représentation d'une structure PLA à NAND.

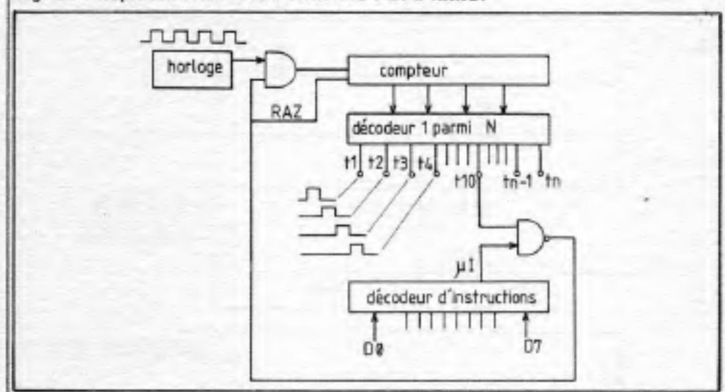


Fig. 36 : Principe d'un distributeur de phase. Si une instruction doit être exécutée en 10 périodes d'horloge, le distributeur sera bloqué à la 10e impulsion grâce à la porte NAND.

suivant la demande en provenance de l'extérieur :

* demande de mise en haute impédance de l'unité centrale
* demande de ralentissement de la fréquence d'horloge (pour des circuits lents)
* changement immédiat de processus pour exécuter en urgence un autre programme (interruptions).

Enfin, et c'est son rôle principal, il doit repérer les micro-instructions dans le temps.

Nous allons faire une analyse succincte de ce séquenceur :

Nous avons vu que l'exécution d'une instruction nécessite un certain nombre de micro-instructions élémentaires. Il est donc très important que chacune de ces micro-instructions, générées par le séquenceur, soit repérée dans le temps ; leurs phases doivent donc être émises par un distributeur de phase dont le principe est présenté fig. 36.

Des impulsions d'horloge sont appliquées à un compteur. Un décodeur 1 parmi N fournit successivement à chacune de ses sorties les phases t1, t2, t3... tn.

Lorsqu'une instruction est décodée, le décodeur d'instruction a une de ses lignes qui passe à 1 ($\mu I = 1$).

Si on considère que cette instruction doit être exécutée en 10 coups d'horloge, il suffira de disposer une porte NAND avec t10 et μI en entrée ; la sortie de la porte bloquant le compteur arrêtera automatiquement le distributeur de phase à la 10e impulsion d'horloge.

Il s'agit maintenant de mettre en équation toutes les micro-instructions de l'unité centrale.

Pour simplifier le processus, supposons que l'unité centrale est dans le cycle exécute de : l'instruction LDA # $3F (charge $3F dans l'accumulateur A : l'instruction LDA est déjà décodée).

Au temps T1 : le PC s'incrémente d'un pas et pointe sur l'opérande (pc + 1 = 1).

t2 : la ligne Read passe à 1 pour lire le contenu de la mémoire = 3F,

t3 : le contenu de pc est placé sur le bus d'adresses pco = 1, Read = 1, on lit donc le contenu pointé par pc,

t4 : l'accumulateur A est chargé (Accin = 1, pco = 1, Read = 1),

t5 : le pc s'incrémente d'un pas (instruction suivante), la ligne Read est désactivée.

L'opérande LDA # $3F est terminée, ce qui nous mène aux équations suivantes :

pc + 1 = LDA.(t1 + t5)
pco = LDA.(t3 + t4)
Read = LDA.(t2 + t3 + t4)
Accin = LDA.t4

et au schéma de la figure 37.

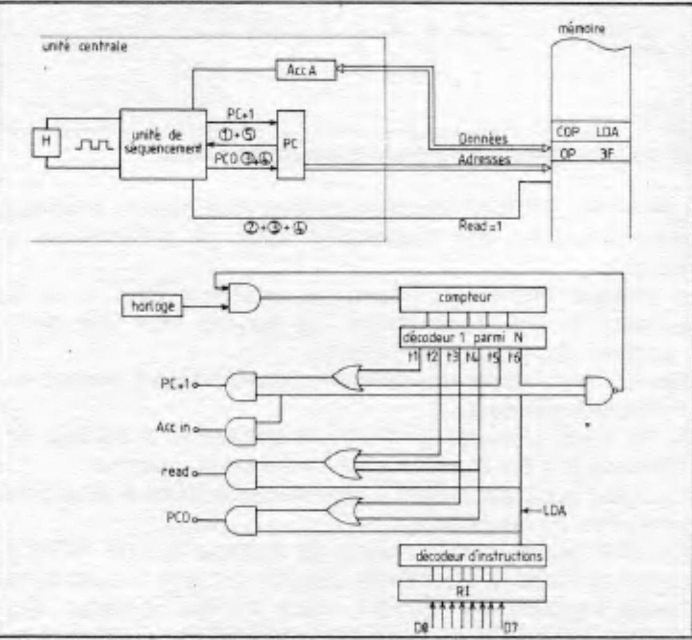


Fig. 37 : Exemple des micro-instructions à générer pour l'instruction LDA (chargement d'une valeur dans l'accumulateur).

Bien sûr, il faudra étudier chacune des micro-instructions pour l'ensemble des instructions.
Par exemple, si nous regardons l'équation de pco pour LDA # $ 12, on trouvera une équivalence avec ADDA # $ 20, CMPA # $ 12, etc.

pco = LDA (t3 + t4)
pco = ADDA (t3 + t4)
pco = CMPA (t3 + t4)

pco peut alors s'écrire pco = (LDA + ADDA + CMPA). (t3 + t4).
On arrive ainsi à un ensemble d'équations représentatives de toutes les micro-instructions de l'unité centrale.
En remplaçant tout cela par un PLA on arrivera à simplifier d'une façon notable le schéma d'un séquenceur.

Le cheminement des informations dans une unité centrale

Nous allons expliquer le cheminement des informations dans une unité centrale.
Il ne s'agira pas de la réalité mais d'une approche car le constructeur ne fournit pas d'explications détaillées sur ce sujet et il faut bien le comprendre !
L'unité centrale doit fournir à l'extérieur au minimum :
- un bus de données bi-directionnel
- un bus d'adresses uni-directionnel
- une ligne de lecture/écriture
- une horloge mettant les périphériques en phase avec le travail de l'unité centrale.

Notons que la ligne READ/WRTIE, les bus de données et d'adresses doivent pouvoir se mettre en haute impédance.
Afin de rendre les micro-instructions totalement transparentes pour le programmeur, on utilisera deux horloges distinctes : une horloge interne (⌀ int) synchronisant les micro-instructions et une horloge externe (f) dont une période représentera le travail de toute une séquence.
Si l'horloge interne a une fréquence quatre fois plus élevée que l'horloge externe, on aura huit périodes d'horloge interne soit huit micro-instructions à générer.
Afin de bien comprendre, nous allons exécuter le programme suivant :

			COP COP
$1000	LDA #	$3F	86 3F
$1002	STA	$5000	B6 50 00
$005			

Soit à charger l'accumulateur A avec la valeur 3F (86 = COP 3F = OP) puis à stocker le contenu de A à l'adresse $5000 (B6 = COP 5000 = 0P). Le programme se trouve localisé à l'adresse $1000.
La figure 38 montre les différentes micro-instructions à générer au sein de l'unité centrale pour ces deux lignes de programme.
Remarquons que l'horloge de 4 MHz fabrique deux signaux : ⌀ int = 125 ns et une période de E = 500 ns.
Les buffers d'adresse, de données et la ligne R/W sont à trois états (signal TSC = Three State Control), le signal latch sert à bloquer les lignes dans un état indéterminé.

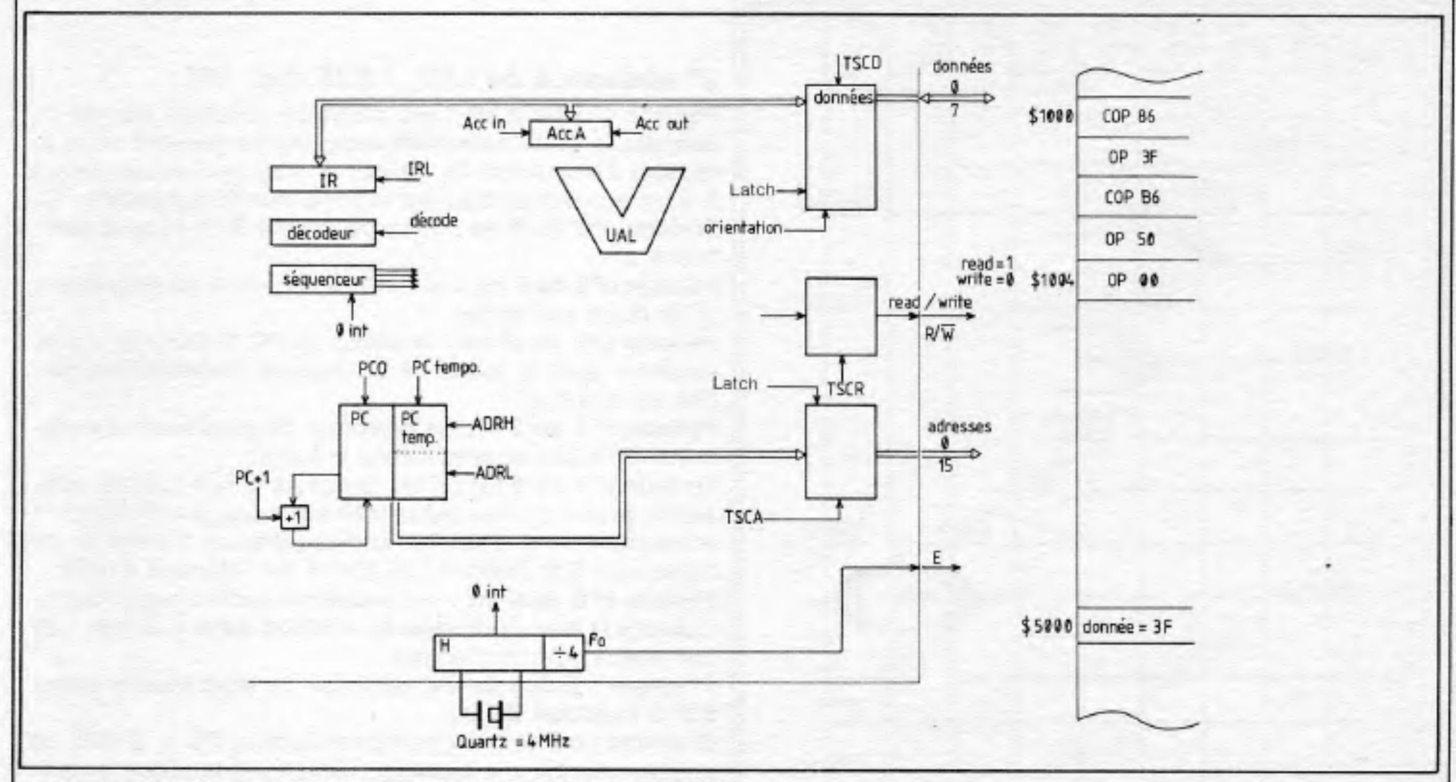


Fig. 38 : Micro-instructions à générer par l'unité centrale pour LDA#$3F et STA $5000.

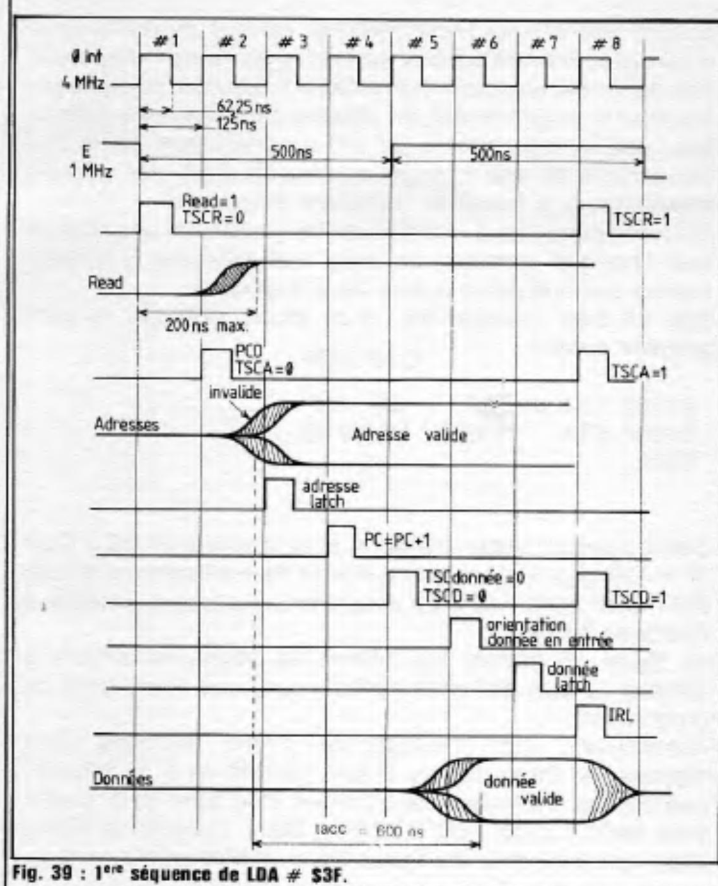


Fig. 39 : 1ère séquence de LDA # $3F.

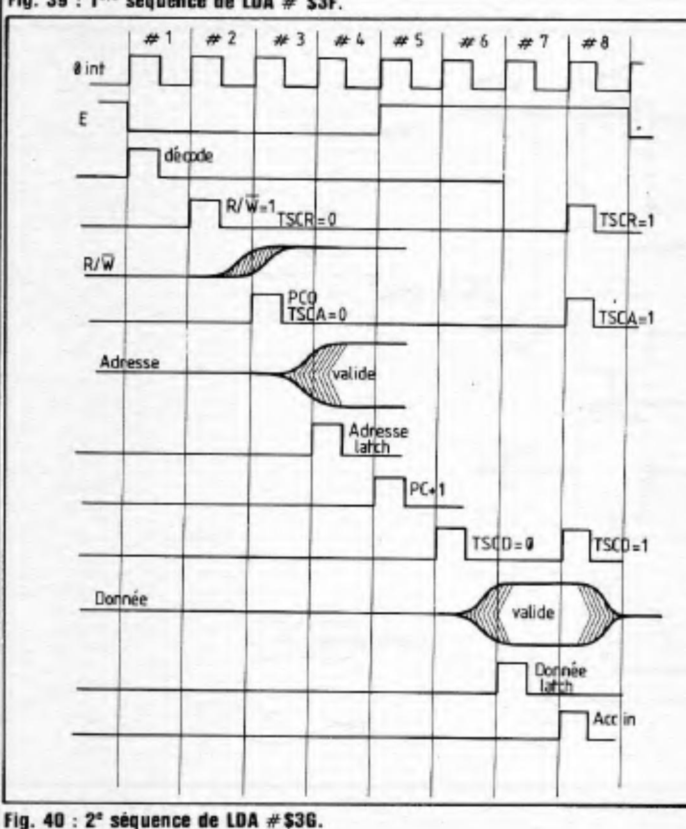


Fig. 40 : 2e séquence de LDA #$3G.

1ere séquence de LDA # $3F (fig. 39) :

On supposera que le compteur de programme PC contient au départ la valeur $ 1000.

Période n°1 de Ø int : on ouvre la ligne R/W (TSRC = 0) et on la positionne à 1, la sortie R/W passe donc à 1 après un certain délai dû à l'établissement de la tension.
Période n°2 de Ø int : on ouvre le compteur de programme (PC output = PCO) ainsi que le buffer d'adresse (TSCA = 0).
Après un certain délai, les adresses sont valides (stables), sur le bus d'adresse.
Période n°3 de Ø int : une impulsion latch sur le buffer d'adresse maintient le bus dans un état stable.
Période n°4 de Ø int : ce qui permet d'incrémenter le compteur de programme (impulsion PC + 1) afin de pointer ultérieurement à l'adresse $1001.
Période n°5 de Ø int : E passe à 1 et y restera jusqu'à la fin de la 8e période de Ø int.
Durant cette période, on ouvre le buffer de données TSCD = 0.
Période n°6 de Ø int : on oriente le buffer de données en entrées (puisqu'on fait une lecture de la mémoire).
Période n°7 de Ø int : la donnée doit être présente sur le bus de données, il suffit donc de la «latcher» dans le buffer.
Période n°8 de Ø int : la donnée étant présente dans le buffer, il suffit de la charger dans le registre d'instruction IR (impulsion Instruction Register Load = IRL) puis de mettre tous les buffers en haute impédance pour finir la séquence TSCR = 1 TSCA = 1 TSCD = 1.

2e séquence de LDA #$3F (fig. 40) :

Période n°1 de Ø int : une impulsion «décode» permet de décoder le code opératoire reçu (86) se trouvant dans le registre d'instruction IR. Il s'agit de charger l'accumulateur A avec la valeur pointée par le compteur de programme PC.
Période n°2 de Ø int : on ouvre la ligne R/W et on la positionne à 1.
Période n°3 de Ø int : on valide le compteur de programme et on ouvre son buffer.
Période n°4 de Ø int : la valeur de PC ($1001) se trouve «latchée» dans le buffer ; à ce moment, l'adresse est validée sur son bus.
Période n°5 de Ø int : le compteur de programme s'incrémente d'un pas et passe donc à $1002.
Période n°7 de Ø int : c'est durant ce temps que l'on considère qu'une donnée est validée sur le bus, il suffit donc de la «latcher» dans le buffer. En l'occurrence, il s'agit ici de l'opérande $3F puisque l'on pointe sur l'adresse $1001.
Période n°8 de Ø int : une impulsion Accin charge l'accumulateur A avec la donnée se trouvant dans le buffer. Les bus sont à nouveau fermés.
7e phase : la donnée est «latchée». On écrit donc la valeur $3F à l'adresse $5000.
8e phase : on ferme les buffers et puisque PC = $1005, on pourra exécuter une nouvelle instruction à la séquence suivante.

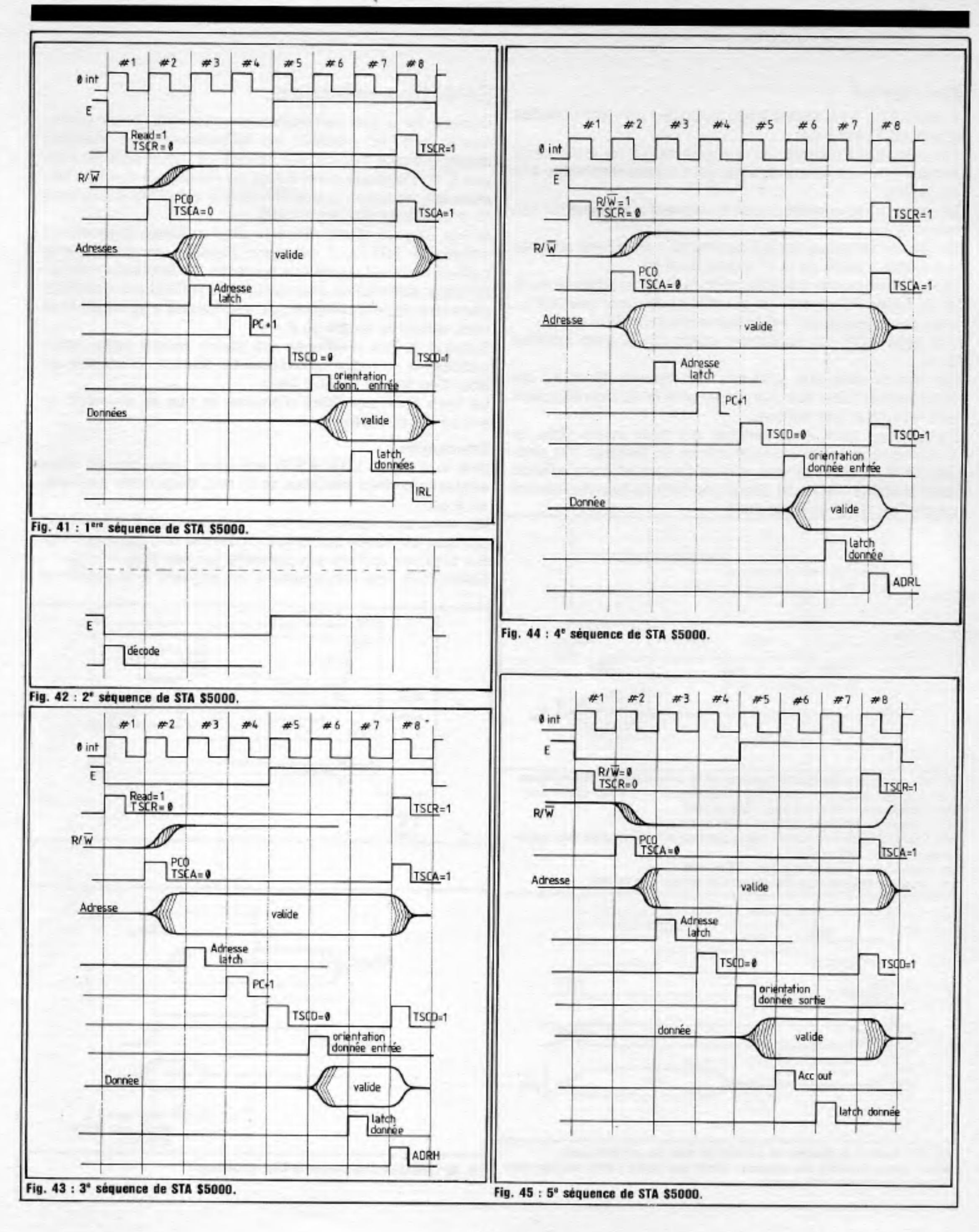


Fig. 41 : 1ère séquence de STA $5000.

Fig. 42 : 2e séquence de STA $5000.

Fig. 43 : 3e séquence de STA $5000.

Fig. 44 : 4e séquence de STA $5000.

Fig. 45 : 5e séquence de STA $5000.

Conclusion

L'instruction STA $5000 s'est exécutée en cinq périodes d'horloge E soit 5 µs.

On remarque à nouveau qu'une adresse est toujours valide lorsque E = 1 et qu'une donnée est toujours disponible à la fin de E = 1.

De plus, l'unité centrale passe finalement la majorité de son temps à lire.

En fait, on remarque sur les figures 39 à 45 qu'une adresse est stable à partir de la 3e phase de Ø int.

Si l'on fabrique une nouvelle horloge Q en quadrature sur E et de même fréquence, on pourrait poser pour postulat :

– qu'une adresse est valide sur le front montant de Q

– la ligne R/W est également valide sur le front montant de Q

On aboutit ainsi aux schémas des figures 46 et 47 qui représentent l'état des bus d'adresses et de données pour une lecture et une écriture.

Remarque : dans les exemples que nous avons cités, le cheminement des micro-instructions en fonction des phases de Ø int n'est qu'une vue de l'esprit en comparaison avec la réalité mais cela donne une idée du fonctionnement interne de toute unité centrale.

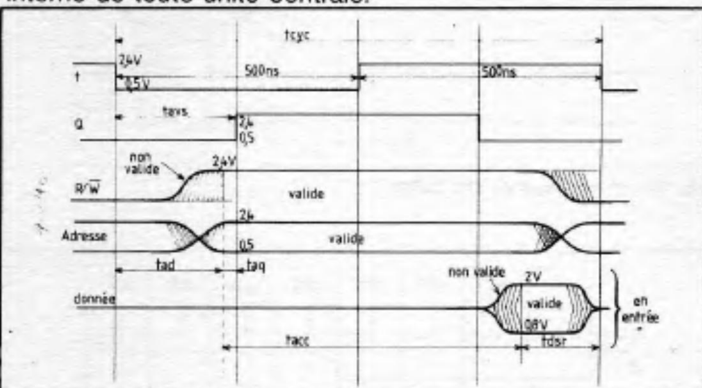


Fig. 46 : Lecture de données en mémoire ou en provenance de périphériques.
tcyc = temps de cycle = durée d'une séquence (si t = 1 MHz – tcyc = 1 µs.)
tavs : temps entre t Bas et Q haut : 250 ns max
tad : temps de retard des adresses : 200 ns max
tacc : temps d'accès à la lecture : temps que mettra l'unité centrale pour acquérir une donnée : 695 ns min.
taq : temps entre adresse et Q haut : 25 ns min.
tds : temps d'établissement des données en lecture : 80 ns min.

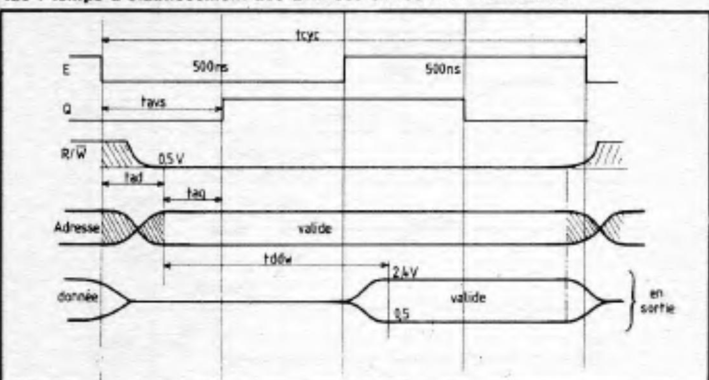


Fig. 47 : Ecriture de données en mémoire ou dans les périphériques.
tddw : temps de retard des données : temps que mettra l'unité centrale pour envoyer une donnée : 225 ns max.

Sélection mémoire

Compte tenu des remarques précédentes, nous remarquons qu'il est possible de sélectionner une mémoire durant la moitié d'un cycle d'horloge de l'unité centrale lorsque E = 1 puisque c'est durant ce moment-là que l'on dispose des adresses et des données (à condition que la ligne R/W soit à un état significatif).

Si l'on dispose d'une mémoire dont le temps d'accès est inférieur à 500 ns, il est donc possible de sélectionner celle-ci comme le montre la figure 48. Un décodeur/démultiplexeur sélectionne une mémoire (une RAM par exemple), dans une plage d'adresses qui correspond à sa capacité et ceci, durant le temps où E = 1.

Puisque le bus d'adresse est stable durant cette demi-période, la mémoire acceptera ou fournira la donnée au bout d'un temps minimal tacc.

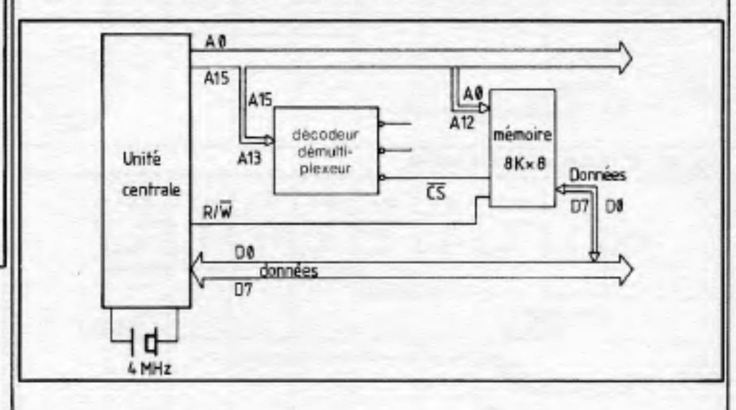
La ligne R/W permettra d'orienter le bus de données en entrée ou en sortie.

Conclusion :

Une instruction LDA #$3F est donc exécutée en deux séquences (deux périodes de E) soit, dans notre exemple, en 2 µs.

On remarque également que c'est pendant E = 1, qu'une adresse est valide sur le bus d'adresse et c'est à la fin de E = 1 qu'une donnée est présente sur son bus.

Continuons nos investigations en passant à la deuxième



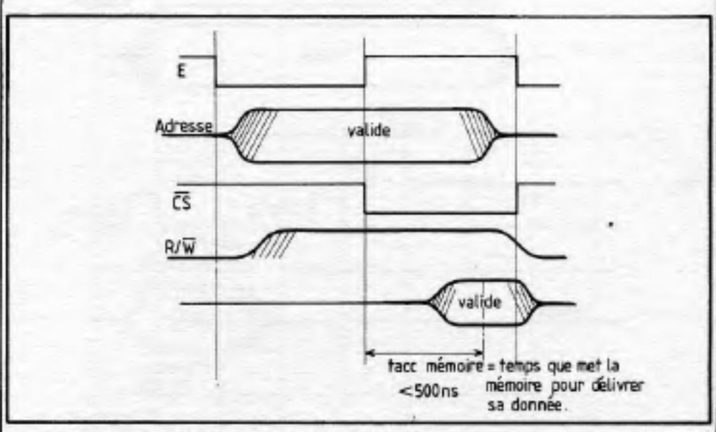


Fig. 48 : Sélection d'une mémoire RAM en lecture.

ligne de programme.
Il s'agit de stocker le contenu de l'accumulateur A à l'adresse $5000.

1ère séquence de STA é5000 (fig. 41) :
Elle ressemble comme une soeur à la première séquence de LDA#$3F, ce qui est normal puisqu'il s'agit d'acquérir le code opératoire (B6) afin de le décoder.

2ème séquence de STA $5000 (fig. 42) :
Il s'agit d'une séquence presque perdue puisqu'elle ne sert qu'à décoder l'instruction.

3ème séquence de STA $5000 (fig. 43) :
Cette séquence est à nouveau classique puisqu'elle ressemble à la première : la donnée reçue ($50 puisque le compteur de programme pointe à l'adresse $1003) est chargée dans la partie haute du registre temporaire d'adresse (ADRH).
Remarquons qu'il ne faut surtout pas charger le compteur de programme avec la valeur $5000 car on perdra définitivement la valeur correspondant à la suite du programme !

4ème séquence de STA $5000 (fig. 44) :
Cette séquence ressemble à la troisième. On charge la donnée ($00 puisque PC pointe en $1004) dans la partie basse du registre temporaire d'adresse (impulsion ADRL). A l'issue de cette séquence, le registre temporaire d'adresses contient la valeur $5000.

5ème séquence de STA $5000 (fig. 45) :
1ère phase : on positionne la ligne R/W à 0 afin d'opérer une écriture en mémoire.
2ème et 3ème phase : on délivre le contenu du registre temporaire sur le bus d'adresses.
4ème et 5ème phase : le buffer de données est orienté dans le sens Unité Centrale-Mémoire.
6ème phase : le contenu de l'accumulateur est délivré sur le bus de données (impulsions Accout).

Le 6809

L'unité centrale que nous allons étudier durant les chapitres qui suivent porte la référence EF 6809, elle est fabriquée par la société Thomson-EFCIS, le créateur étant Motorola.
Ce microprocesseur existe en trois versions :
- EF 6809 fonctionne avec une fréquence d'entrée de 4 MHz, et délivre donc une fréquence E = 1 MHz ;
- EF 68 A 09 avec f = 6 MHz et E = 1,5 MHz ;
- EF 68 B 09 avec f = 8 Mhz et E = 2 MHz.

La puissance dissipée est de 1 W max. (sous 5 V).

La figure 49b nous donne le brochage de ce composant, on note qu'il comprend :

Le 6809 comprend cinq registres internes de 16 bits et 4 registres de 8 bits (fig. 49a), ils seront décrits en détail dans le chapitre suivant.
Nota : la suite de ce chapitre est tirée de documents EFCIS.

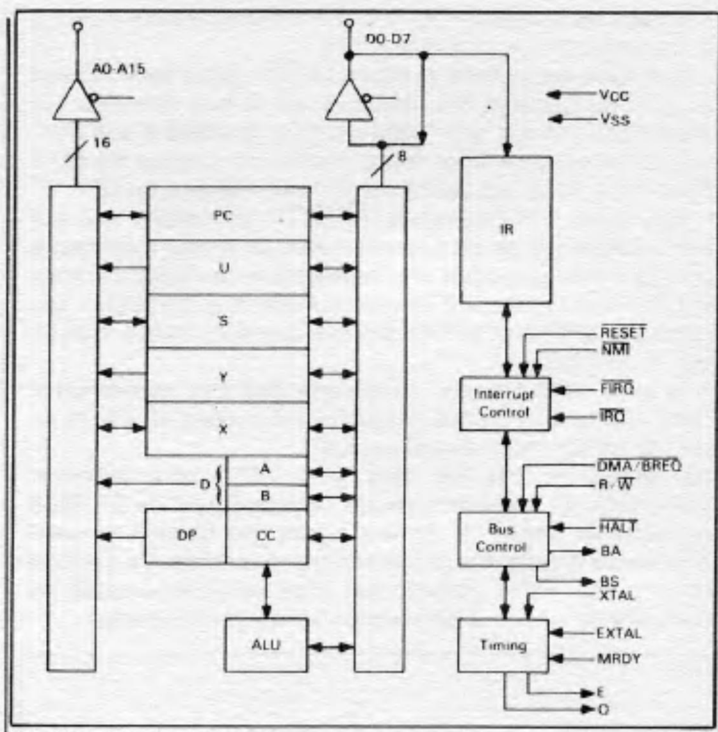


Fig. 49a : Synoptique du 6809.

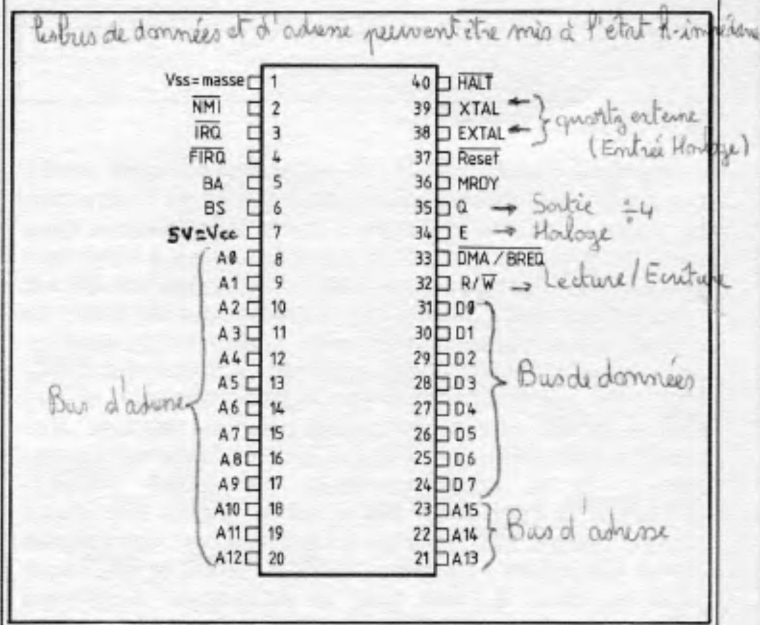


Fig. 49b : Brochage du 6809.

- **un bus d'adresses** (A0-A15) : lorsque le bus n'est pas utilisé par le 6809 pour un transfert de données, il sort l'adresse $FFFF R/W = 1 et BS = 0.

Nous avons vu que les adresses sont validées sur le front montant de Q. Tous les amplificateurs du bus d'adresses sont mis à l'état haute impédance lorsque la sortie BA est à l'état haut.

- **un bus de données** (D0-D7) : il est bidirectionnel et sert à la transmission de données 8 bits ;
- **une ligne de lecture-écriture** (R/$\overline{W}$) : cette ligne indique le sens du transfert des données sur le bus données. Un niveau bas indique que l'unité centrale procède à une écriture, R/W passe à l'état haute impédance lorsque BA est à l'état haut. R/W est validé sur le front montant de Q.
- **une ligne d'initialisation**($\overline{RESET}$) : un niveau bas sur cette entrée trigger de Schmitt durant un temps supérieur à un cycle bus provoque une initialisation de l'unité centrale (fig. 50). Les vecteurs d'initialisation seront accessibles aux adresses \$FFFE et \$FFFF dès lors que la condition logique BA = 0 et BS = 1.

A la mise sous tension, cette ligne doit être maintenue à l'état bas jusqu'à ce que l'oscillateur d'horloge ait atteint un régime de fonctionnement normal.

Un simple réseau RC peut être utilisé pour initialiser l'ensemble du système puisque l'entrée Reset du EF 6809 possède un trigger de Schimtt ayant une tension de seuil supérieure à celle des périphériques standards. Ce seuil de tension plus élevé garantit que tous les périphériques ne sont pas en phase d'initialisation après le processeur.

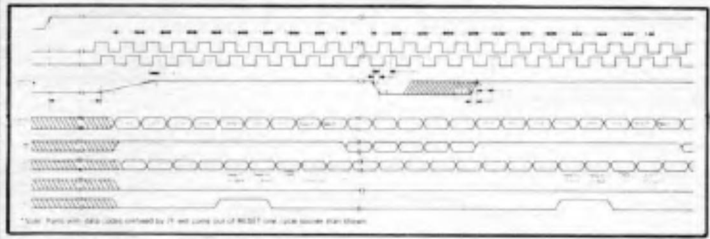


Fig. 50 : Fonctionnement d'un RESET.

- **une ligne d'arrêt** ($\overline{HALT}$) : un niveau bas sur cette entrée provoque l'arrêt du microprocesseur à la fin de l'instruction en cours et celui-ci demeure à l'arrêt **indéfiniment** sans perte de données. A l'arrêt, la sortie BA passe à l'état haut indiquant que les bus sont à l'état haute impédance. BS est aussi à l'état haut indiquant que le processeur est arrêté ou à l'état bus accordé. A l'état arrêt, le microprocesseur ne répond **pas à des** demandes externes en temps réel ($\overline{FIRQ}$, $\overline{IRQ}$) bien que $\overline{DMA}/\overline{BREQ}$ soit toujours accepté, et que $\overline{NMI}$ et $\overline{RESET}$ soient mémorisées pour une réponse ultérieure. A l'état arrêt, Q et E continuent à fonctionner normalement. Si le microprocesseur est arrêté ($\overline{RESET}$, $\overline{DMA}/\overline{BREQ}$), l'état $\overline{HALT}$ (BA et BS = 1) peut être atteint lorsque l'entrée $\overline{Halt}$ est mise à l'état bas bien que l'entrée Reset soit encore à l'état bas. Si $\overline{DMA}/\overline{BREQ}$ et $\overline{HALT}$ sont tous les deux à l'état bas, le processeur continuera jusqu'au dernier cycle de l'instruction sur lequel le processeur sera arrêté (fig. 51).

- **deux lignes d'état du bus** (BA = Bus Available, BS = Bus Status : BA = Bus accordé ou Bus libre, BS = état du bus.

La sortie BA indique qu'un signal de commande interne fait passer les bus du microprocesseur à l'état haute impédance. Ce signal n'implique pas que le bus soit disponible pendant plus d'un cycle.

Lorsque BA passe à l'état bas, un cycle perdu supplémentaire se déroule avant que le microprocesseur n'occupe le bus.

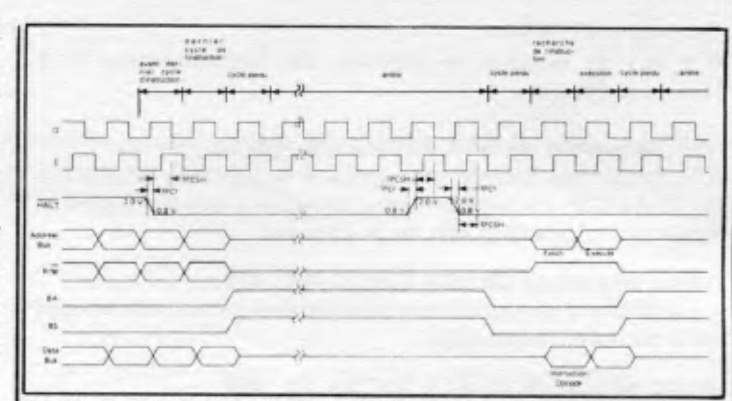


Fig. 51 : Fonctionnement du halt et exécution d'une seule instruction.

Le signal de sortie état du bus, lorsqu'il est décodé avec BA, représente l'état du microprocesseur (validé sur le front montant de Q) :

BA	BS	Etat du microprocesseur
0	0	Fonctionnement normal
0	1	Reconnaissance d'interruptions
1	0	Reconnaissance de l'instruction SYNC
1	1	Arrêt ou Bus accordé

Une reconnaissance d'interruption (BA = 0, BS = 1) est présentée durant les deux cycles d'acquisition du vecteur d'interruption ($\overline{RESET}$, $\overline{NMI}$, $\overline{IRQ}$, SWI, SW12, SW13).

Cet état détecté et le décodage des quatre lignes d'adresses de poids faibles indiquent à l'utilisateur quel est le niveau d'interruption pris en compte et permet une vectorisation par les périphériques.

Il y a reconnaissance de synchronisation lorsque le microprocesseur rencontre l'instruction de synchronisation (SYNC), celle-ci est indiquée par BA = 1 et BS = 0, signifiant que le microprocesseur est en attente de synchronisation extérieure par l'intermédiaire d'une ligne d'interruption.

La condition BA = BS = 1 est vraie lorsque le microprocesseur est dans l'état Halt ou bus accordé (ligne $\overline{Halt}$ = 0).

- **une ligne d'interruption Non Masquable** ($\overline{NMI}$ = Non Masquable Interrupt) : un front descendant sur cette entrée entraîne une séquence d'interruption non masquable. Une interruption non masquable ne peut pas être inhibée par programme et possède une priorité supérieure à $\overline{FIRQ}$, $\overline{IRQ}$ ou aux interruptions logicielles. Lors d'une reconnaissance de $\overline{NMI}$, l'état complet du microprocesseur est sauvegardé sur la pile. Après initialisation, une $\overline{NMI}$ ne sera prise en compte qu'après le premier chargement par programme du pointeur de pile S.

La largeur d'impulsion de $\overline{NMI}$, à l'état bas, doit être au moins d'un cycle E. Si l'entrée $\overline{NMI}$ n'a pas un temps d'établissement suffisant en regard de Q, l'interruption ne sera prise en compte qu'au cycle suivant (fig. 52).

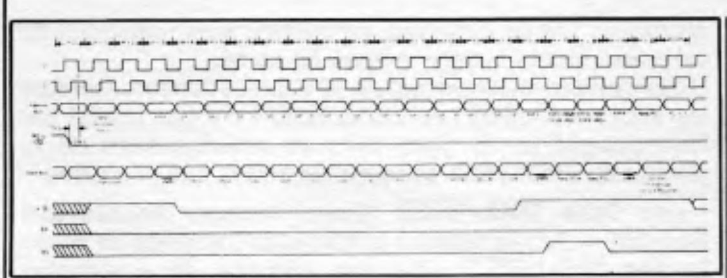


Fig. 52 : Exécution de $\overline{IRQ}$ ou $\overline{NMI}$.

– une ligne de demande d'interruption rapide ($\overline{FIRQ}$ = Fast Interrupt Request) : un niveau bas sur cette broche entraîne la séquence d'interruption rapide, à condition que le bit masque F du registre de condition soit à 0.
Cette interruption a priorité par rapport à une demande d'interruption standard $\overline{IRQ}$, elle est plus rapide puisqu'il n'y a sauvegarde sur la pile que du registre code condition et du compteur de programme. Le sous-programme de traitement des interruptions doit libérer la source d'interruption avant l'exécution de l'instruction RTI (fig. 53).

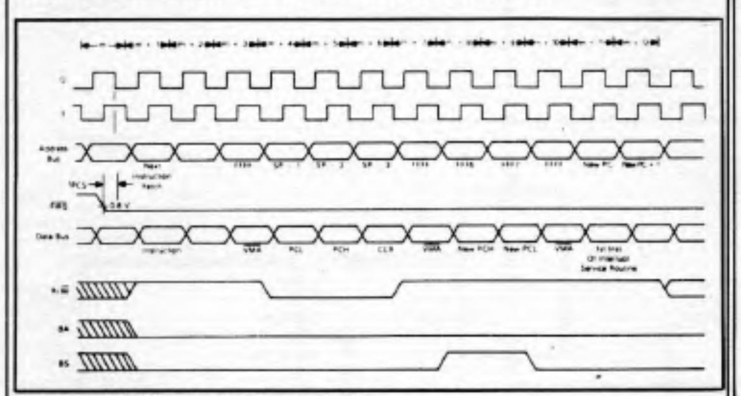
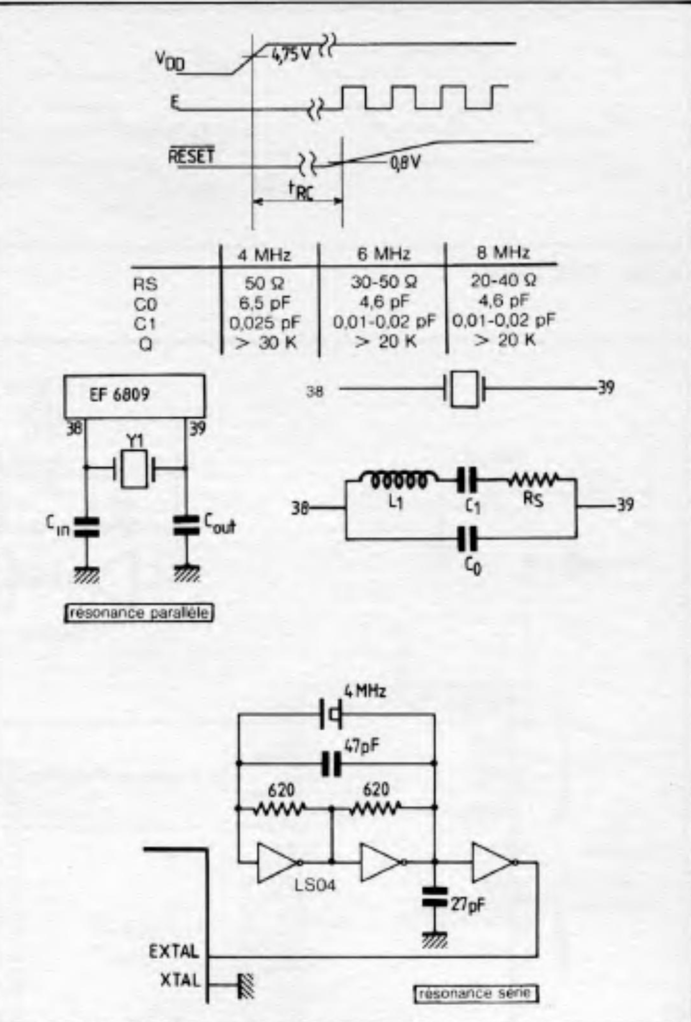


Fig. 53 : Exécution de l'interruption FIRQ.

– une ligne de demande d'interruption (IRQ = Interrupt Request) : un niveau bas appliqué à cette entrée entraîne la séquence de traitement d'interruption $\overline{IRQ}$, à condition que le bit masque I du registre de condition soit à 0.
Cette séquence réalisant la sauvegarde de l'état complet du processeur, la réponse sera plus lente que pour le FIRQ. Par ailleurs, IRQ a une priorité plus basse que FIRQ. Là encore, le sous-programme de traitement des interruptions doit libérer la source d'interruption avant d'exécuter l'instruction RTI (fig. 52).

– deux lignes XTAL et EXTAL : ces broches d'entrée sont utilisées pour connecter l'oscillateur interne à un quartz externe à résonance parallèle.
Par ailleurs, la broche EXTAL peut être utilisée comme une entrée niveau TTL pour une horloge extérieure en mettant XTAL à la masse. Le quartz ou la fréquence externe est quatre fois la fréquence bus (fig. 54). Les règles d'implantation propres aux circuits RF doivent être observées dans le tracé des circuits imprimés.

– deux lignes E et Q : E est identique au signal d'horloge ⌀ 2 du EF 6800 ; Q est un signal d'horloge en quadrature qui pilote E.



	4 MHz	6 MHz	8 MHz
RS	50 Ω	30-50 Ω	20-40 Ω
C0	6,5 pF	4,6 pF	4,6 pF
C1	0,025 pF	0,01-0,02 pF	0,01-0,02 pF
Q	> 30 K	> 20 K	> 20 K

Fig. 54 : Connexion du quartz et démarrage de l'oscillateur.

Q n'a pas d'équivalent sur le EF 6800. Les adresses du microprocesseur sont validées sur le front montant de Q. Les données sont mémorisées sur le front descendant de E.
Le diagramme des temps pour E et Q est montré figure 55.

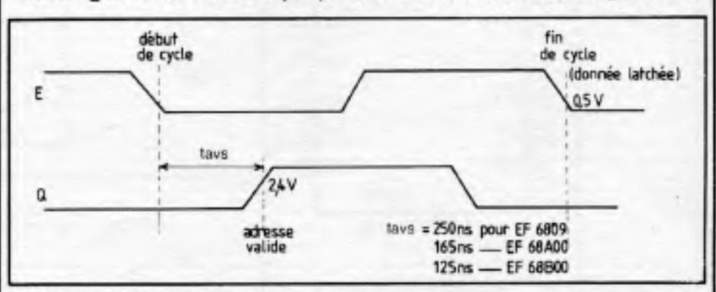


Fig. 55 : Relation entre t et Q.

– une ligne Memory Ready (MRDY = mémoire prête) : cette entrée de commande permet l'allongement de E pour augmenter le temps d'accès aux données. Lorsque MRDY est à l'état haut, E est en fonctionnement normal. Lorsque MRDY est à l'état bus, E peut être allongé de multiples

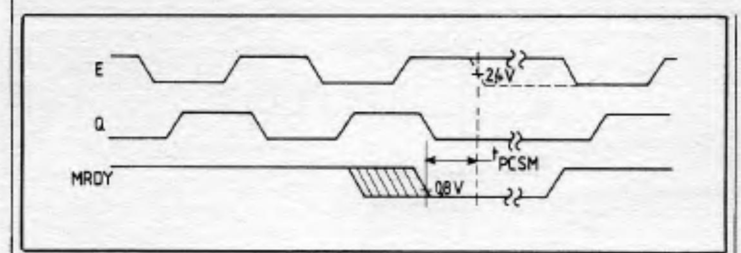


Fig. 56a : MRDY Timing.

entiers de 1/4 de cycle bus, permettant ainsi l'utilisation de mémoires lentes comme indiqué figure 56.
L'allongement maximum est de 10 microsecondes. Pendant les accès mémoires non utiles, MRDY n'a pas d'effet sur l'allongement de E. Ceci évite le ralentissement de la vitesse du processeur pendant les accès bus non utiles.

– **une ligne $\overline{\text{DMA}}/\overline{\text{BREQ}}$** (Direct Memory Access/Bus Request = Accès Direct à la Mémoire/Demande de Bus) : l'entrée $\overline{\text{DMA}}/\overline{\text{BREQ}}$ offre une méthode de suspension d'exécution et d'acquisition du bus du microprocesseur

Fig 56b : MRDY synchronisation.

VALEURS LIMITES

Valeurs	Symboles	Valeurs	Unités
Tension d'alimentation	V_{CC}	−0.3 à +7.0	V
Tension d'entrée	V_{in}	−0.3 à +7.0	V
Température de fonctionnement	T_A	0 à +70	°C
Température de stockage	T_{stg}	−55 à +150	°C
Résistance thermique	θ_{JA}	70	°C/W

Les entrées de ce circuit sont protégées contre les hautes tensions statiques et les champs électriques ; toutefois, il est recommandé de prendre les précautions normales pour éviter toute tension supérieure aux valeurs limites sur ce circuit à haute impédance.

CARACTÉRISTIQUES ÉLECTRIQUES (V_{CC} = 5,0 V ± 5 %, V_{SS} = 0, T_A = 0 à 70°C sauf spécifications contraires)

Caractéristiques		Symboles	Min	Typ	Max	Unités
Tension d'entrée à l'état haut	logique, EXtal $\overline{RESET}$	V_{IH}	V_{SS} + 2.0 V_{SS} + 4.0	— —	V_{DD} V_{DD}	V
Tension d'entrée à l'état bas	logique, $\overline{RESET}$, EXtal	V_{IL}	V_{SS} - 0.3	—	V_{SS} + 0.8	V
Courante de fuite en entrée (V_{in} = 0 à 5,25 V, V_{CC} = max)	logique	I_{in}	—	1.0	2.5	µA
Tension de sortie à l'état haut (I_{charge} = −205 µA, V_{CC} = min) (I_{charge} = −145 µA, V_{CC} = min) (I_{charge} = −100 µA, V_{CC} = min)	 D0-D7 A0-A15, $R/\overline{W}$, Q, E BA, BS	V_{OH}	 V_{SS} + 2.4 V_{SS} + 2.4 V_{SS} + 2.4	 — — —	 — — —	V
Tension de sortie à l'état bas (I_{charge} = 2,0 mA, V_{CC} = min)		V_{OL}	—	—	V_{SS} +0.5	V
Puissance dissipée		P_D	—	—	1.0	W
Capacité # (V_{in} = 0, T_A = 25°C, f = 1,0 MHz)	 D0-D7 entrées logiques, EXtal A0-A15, $R/\overline{W}$	C_{in} C_{out}	 — — —	 10 7 —	 15 10 12	pF
Fréquence de travail (Quartz ou entrée extérieure)	EF6809 EF68A09 EF68B09	f f_{XTAL} f_{XTAL}	— — —	— — —	4 6 8	MHz
Trois états, courant d'entrée (V_{in} = 0,4 à 2,4 V, V_{CC} = max)	D0-D7 A0-A15, $R/\overline{W}$	I_{TSI}	— —	2.0 —	10 100	µA

CARACTERISTIQUES DYNAMIQUES (lecture - écriture) (figures 1 et 2)

Caractéristiques	Symboles	EF6809			EF68A09			EF68B09			Unités
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
Temps de cycle	t_{CYC}	1000	—	—	667	—	—	500	—	—	ns
Temps utile total	t_{UT}	975	—	—	640	—	—	480	—	—	ns
Temps d'accès à la lecture t_{ac} = (t_{AD} - t_{DSR})	t_{ACC}	695	—	—	440	—	—	320	—	—	ns
Temps d'étab. données (lect.)	t_{DSR}	80	—	—	60	—	—	40	—	—	ns
Temps de maintien donn. (ent.)	t_{DHR}	10	—	—	10	—	—	10	—	—	ns
Temps de maintien donn. (sort.)	t_{DHW}	30	—	—	30	—	—	30	—	—	ns
Temps maintien des adresses (adresses, $R/\overline{W}$)	t_{AH}	30	—	—	30	—	—	30	—	—	ns
Temps de retard des adresses	t_{AD}	—	—	200	—	—	140	—	—	110	ns
Temps de retard des données	t_{DDW}	—	—	225	—	—	180	—	—	145	ns
Temps entre E bas et Q haut	t_{AVS}	—	—	250	—	—	165	—	—	125	ns
Temps entre adresse et Q haut	t_{AQ}	25	—	—	25	—	—	15	—	—	ns
Horloge proces., état bas	t_{PWEL}	450	—	—	295	—	—	210	—	—	ns
Horloge proces., état haut	t_{PWEH}	450	—	—	280	—	—	220	—	—	ns
MRDY, temps d'établissement	t_{PCSR}	60	—	—	60	—	—	60	—	—	ns
Temps d'établis., interruptions	t_{PCS}	200	—	—	140	—	—	110	—	—	ns
$\overline{HALT}$, temps d'établissement	t_{PCSH}	200	—	—	140	—	—	110	—	—	ns
$\overline{RESET}$, temps d'établissement	t_{PCSR}	200	—	—	140	—	—	110	—	—	ns
$\overline{DMA/BREQ}$, temps d'établis.	t_{PCSD}	125	—	—	125	—	—	125	—	—	ns
Temps démarrage de l'oscillateur	t_{rc}	100	—	—	100	—	—	100	—	—	ms
E, tps de montée et de descente	t_{ER}, t_{EF}	5	—	25	5	—	25	5	—	20	ns
Temps de montée et de descente du processeur	t_{PCR}, t_{PLF}	—	—	100	—	—	100	—	—	100	ns
Q, tps de montée et de descente	t_{QR}, t_{QF}	5	—	25	5	—	25	5	—	20	ns
Q, état haut	t_{PWQH}	450	—	—	280	—	—	220	—	—	ns

pour une autre utilisation comme montré en figure 57.
Des utilisations types comprennent le DMA et le rafraîchissement des mémoires dynamiques.
La transition de DMA/BREQ doit se produire pendant Q. Un niveau bas sur cette broche arrêtera l'exécution de l'instruction à la fin du cycle en cours.
BA = BS = 1 indique la prise en compte de la demande faite par DMA/BREQ, le circuit demandeur aura alors jusqu'à 15 cycles bus avant que le microprocesseur ne récupère le bus pour auto-rafraîchissement.
L'auto-rafraîchissement nécessite un cycle bus comportant un cycle perdu de début et de fin (figure 57).
En général, le contrôleur de DMA fait une demande d'accès au bus en mettant au niveau bas la broche DMA/BREQ sur le front montant de E. Lorsque le microprocesseur répond avec BA = BS = 1, ce cycle est un cycle perdu utilisé pour transférer le contrôle au système de DMA.

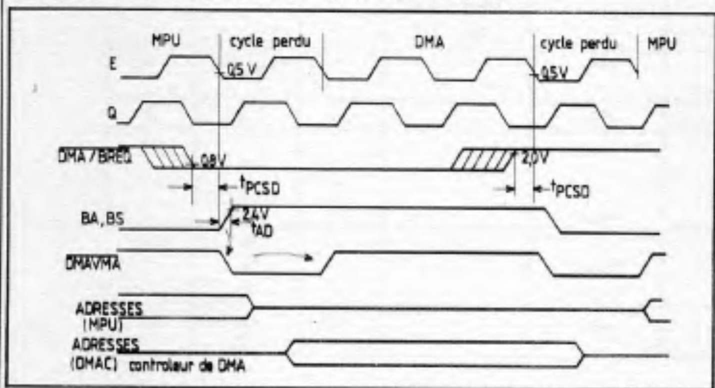


Fig. 57a : Diagramme de temps typique de l'entrée DMA (< 14 cycles).

Les faux accès mémoires doivent être évités pendant tout cycle perdu. Lorsque BA est remis à 0 (soit comme résultat de DMA/BREQ = 1, ou auto-rafraîchissement du microprocesseur), le circuit DMA doit être déconnecté du bus. Un autre cycle perdu s'écoule avant que le microprocesseur ne se voit alloué un accès mémoire pour transférer le contrôle sans litige.

Nota : se reporter au chapitre III pour de plus amples informations sur les lignes RESET, NMI, FIRQ, IRQ, Halt, DMA/BREQ.

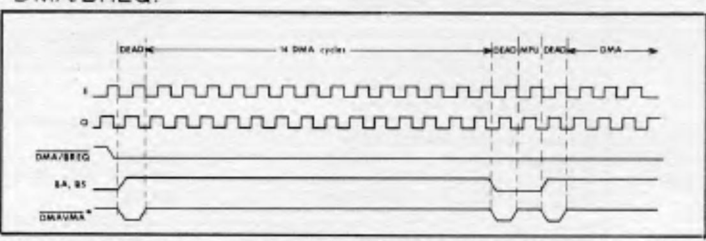


Fig. 57b : Diagramme des temps auto-rafraîchissement en DMA (> 14 cycles).

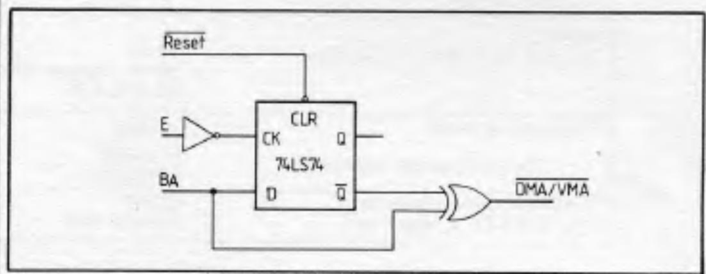


Fig. 57c : Logique de DMA/VMA.

Fig. 57. DMA/VMA est un signal externe au CPU, mais nécessaire pour la gestion du DMA.

Chapitre 2

Un petit système d'initiation : le Microkit 09

DESCRIPTION DE LA CARTE UNITE CENTRALE ET DE LA CARTE CLAVIER.

Description de la carte unité centrale (U.C.)

La carte unité centrale est le cœur et le «cerveau» du Micro Kit 09*. C'est elle qui supporte le microprocesseur 6809, sa mémoire vive, sa mémoire morte ainsi qu'un périphérique d'entrée-sortie (6821).

Dans un système à microprocesseur, il faut, pour bien utiliser toutes ses possibilités, connaître les deux points suivants :
- L'architecture Hard (matériel) du système. (Fonctionnement des différents éléments constituant la ou les cartes).
- Maîtriser le Soft (logiciel) c'est-à-dire bien connaître toutes les possibilités du microprocesseur au niveau des instructions.

Le deuxième point sera expliqué en détail dans la troisième partie de cet ouvrage. Pour le moment, nous allons nous intéresser à la partie matériel du Micro Kit 09.
La carte unité centrale est donc composée de cinq circuits intégrés (6809, 6821, 6116, 74LS 138, 2716) dont nous allons détailler dans la suite le brochage et le fonctionnement.

Le CPU (Control Processing Unit) 6809. Description des broches

Le 6809 est un circuit intégré en boîtier DIL 40 broches fabriqué en technologie MOS canal N et mono-tension 5 V. Le brochage de ce microprocesseur est donné en figure 1.

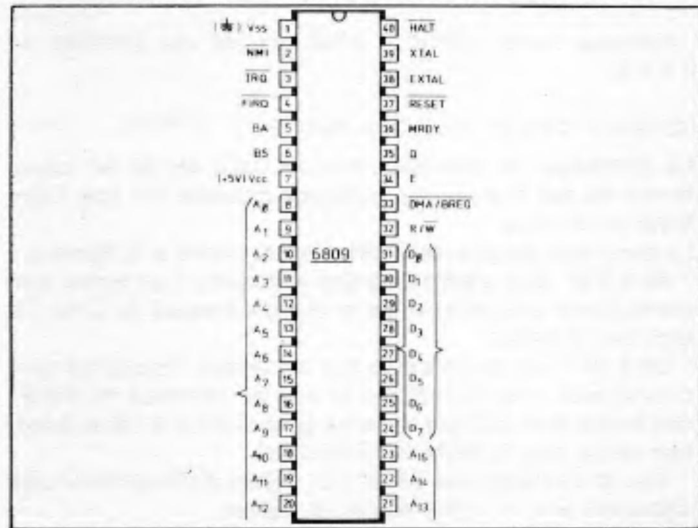


Fig. 1 : Le microprocesseur 6809.

* Le Micro Kit 09 est un micro-ordinateur simplifié permettant l'étude du microprocesseur 6809.

– Ao → A15 (broches 8 à 23)
Bus d'adresses du microprocesseur. C'est un bus de 16 bits, ce qui permet un adressage de 65 536 (2^{16}) cases mémoires (mémoires ou périphériques). Ces lignes sont le reflet du Registre Compteur Programme (CP) du microprocesseur. Ce sont des sorties.

– Do → D7 (broches 24 à 31)
Bus de données du microprocesseur. C'est un bus bidirectionnel de 8 bits par lequel transitent toutes les informations de la périphérie vers le microprocesseur et vice-versa.

– Vss et Vcc (broches 1 et 7)
Broches d'alimentation du boîtier : Vss est la masse et Vcc le +5 V.

– $\overline{NMI}$, $\overline{IRQ}$; $\overline{FIRQ}$ (broches 2, 3 et 4)
Ce sont les lignes d'interruption du 6809. Lorsqu'un niveau logique 0 est présent sur l'une de ces entrées, le programme en cours d'execution est interrompu et le microprocesseur exécute un programme dit d'interruption (voir chapitre 3).

$\overline{NMI}$:	ligne d'interruption non masquable
$\overline{IRQ}$:	ligne d'interruption la moins prioritaire
FIRQ :	ligne d'interruption rapide

Ces lignes sont toutes des entrées.

– BA ; BS (broches 6 et 5) : ce sont les lignes d'état du microprocesseur. Ce sont des sorties et le tableau 1 donne l'état du microprocesseur en fonction de BA et BS.

Tableau n°1

BA	BS	Etat du microproceseur
0	0	Normal
0	1	Reconnaissance d'interruption
1	0	Reconnaissance de synchro. externe
1	1	Arrêt ou Bus Accordé.

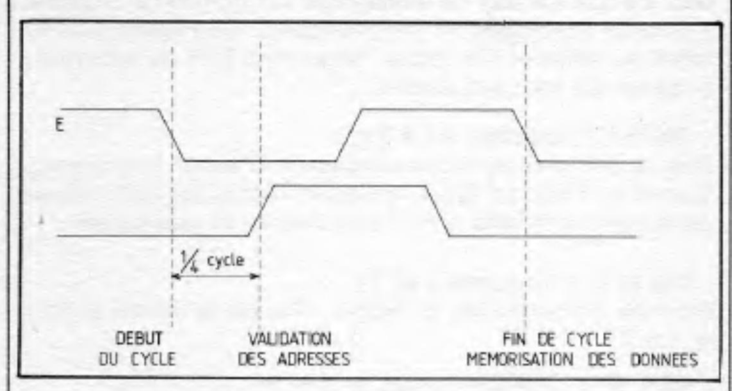
– R/$\overline{W}$ (broche 32) : ligne de lecture/écriture. Cette ligne indique à la périphérie si le microprocesseur est en lecture ou en écriture de données. Cette ligne est une sortie.

$R/\overline{W}$ = 1 → lecture, le bus de données est en entrée
$R/\overline{W}$ = 0 → écriture, le bus de données est en sortie

– E et Q (broches 34 et 35) : signaux d'horloge du CPU.
E : signal de synchronisation avec la périphérie. La fréquence de cette ligne est celle de base du microprocesseur (horloge divisée par 4).

Q : ce signal est en quadrature avec E.

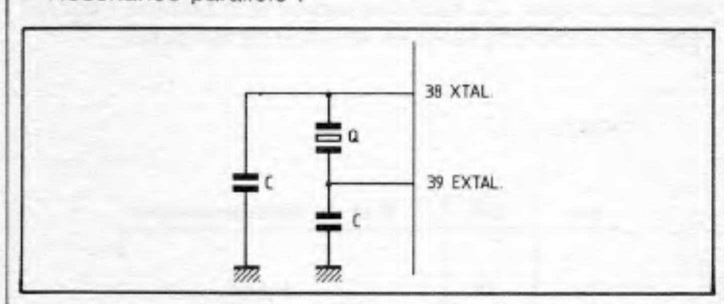
Les adresses des microprocesseurs sont correctes à partir d'un front montant de Q. Les données sont mémorisées sur un front descendant de E. Ces deux lignes sont des sorties.



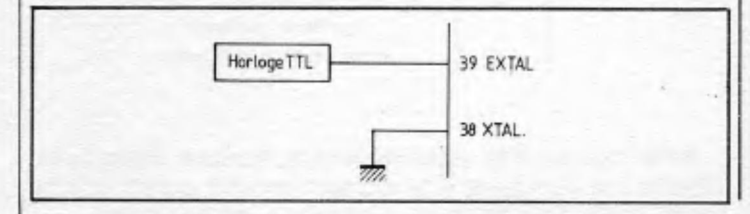
– Halt (broche 40) : cette ligne permet d'interrompre un programme en cours en appliquant un niveau bas sur cette entrée.
Le microprocesseur termine l'instruction en cours puis arrête le déroulement du programme.
Lorsque cette ligne repasse au niveau haut, le programme se continue sans qu'il y ait perte d'information.

– XTAL et EXTAL (broches 38 et 39) : ces broches sont connectées au circuit horloge du CPU. Deux schémas peuvent être adoptés pour ces entrées :

– Résonance parallèle :



– Possibilité de brancher une horloge TTL externe :



Le quartz ou l'horloge externe sont égales à quatre fois la fréquence bus (E et Q). Le schéma choisi pour le micro-kit est le premier avec Q = 4 MHz et C = 22 pF.

– $\overline{\text{RESET}}$ (broche 37) : broche d'initialisation Hard du microprocesseur. Cette ligne active au niveau bas a pour effets :
. arrêt de l'instruction en cours
. registre de page DP = 00
. interruptions $\overline{\text{IRQ}}$ et $\overline{\text{FIRQ}}$ sont masquées
. l'interruption non masquable $\overline{\text{NMI}}$ est désarmée
Le vecteur d'initialisation est disponible aux adresses FFFE et FFFF, c'est-à-dire que le microprocesseur dépose dans un premier temps l'adresse FFFE sur le bus d'adresses et lit un premier octet que nous appellerons octA, puis dépose FFFF et lit un second octet que nous appellerons octB. L'adresse formée par octA et octB est celle de la première instruction exécutée par le microprocesseur.
Cette ligne est une entrée.
(Toute mise sous tension commence automatiquement par un RESET.)

– MRDY (broche 36) : cette broche permet par un allongement de l'horloge E du CPU, d'autoriser la lecture ou l'écriture d'une mémoire ou d'un périphérique dont le temps d'accès est plus lent que celui du 6809; La valeur maximale est de 10 µs. Cette ligne est une entrée.

– $\overline{\text{DMA/BREQ}}$ (broche 33) : cette ligne permet une utilisation des bus du microprocesseur par un circuit extérieur (mise en haute impédance des bus). Par exemple, pour faire du DMA ou du rafraîchissement mémoire.
Ainsi s'achève la description des broches du microprocesseur.

Brochages des mémoires de la carte CPU

Les mémoires utilisées sont les suivantes :
- mémoire vive : RAM 6116 qui est une ram de 2 K × 8 contenant chacune un mot de 8 bits.

(Random Access Memory) = RAM

2048 adresses = 2K

- mémoire morte : EPROM 2716 qui est une EPROM de 2 K×8.

(Erasable program read only memory) = EPROM.

Le brochage de ces deux circuits sera étudié en même temps du fait que seules quelques broches ont des fonctions différentes.
Le brochage de ces deux circuits est donné à la figure 2.
– A0 à A10 : bus d'adresses des mémoires. Ces lignes sont connectées directement sur le bus d'adresses du CPU. Ce sont des entrées.
– D0 à D7 : bus de données des mémoires. Ces lignes sont connectées directement sur le bus de données du 6809 ; ces lignes sont bidirectionnelles pour la RAM 6116 et à lecture seule pour la ROM 2716 (sorties).
– Vcc et Vss (broches 24 et 12) : lignes d'alimentation des mémoires Vcc = +5 V et Vss = masse.

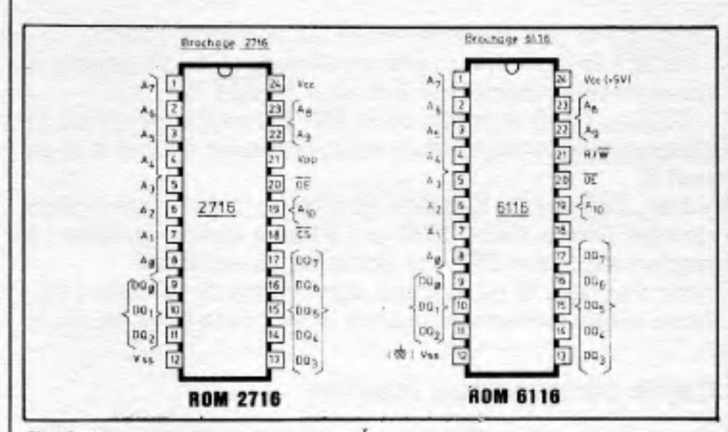


Fig. 2

- $\overline{CS}$ (broches 18) ou Chip Select : broche de sélection du boîtier. Un niveau bas sur cette entrée signale au circuit considéré que le microprocesseur désire dialoguer avec lui. Cette broche a pour effet de passer le bus de données des mémoires dans un état haute impédance lorsqu'elle est au niveau 1 logique (+5 V).
- $\overline{OE}$ (broche 20) ou Output Enable : lorsqu'elle est à 1, cette ligne permet une diminution de la consommation du boîtier pour alimenter, par exemple, le boîtier avec une pile (sauvegarde de la RAM alimentation coupée). Au niveau bas, cette ligne a le même effet que le CS.

Pour sélectionner un boîtier, il faut donc que les broches $\overline{CS}$ et $\overline{OE}$ soient toutes les deux au 0 logique (voir décodage d'adresse).
- R/$\overline{W}$ (broche 21 de 6116) : ligne d'écriture ou de lecture de la RAM. Cette ligne donne le sens des données :
R/$\overline{W}$ = 1 → D0 à D7 de la RAM en sortie.
R/$\overline{W}$ = 0 → D0 à D7 de la RAM en entrée.
La ROM 2716 ne possède pas cette ligne du fait qu'elle est à lecture seule.
- Vpp (broche 21 de la 2716) : cette ligne est utilisée pour programmer la mémoire 2716. En mode normal, cette ligne est au +5 V. En mode programme, cette ligne est à +25 V. Pour le Micro Kit cette ligne est reliée en permanence au +5 V.

Circuit de décodage d'adresse de la carte CPU (74LS138)

Ce circuit est un multiplexeur démultiplexeur 1 parmi 8 dont le brochage est donné figure 3.

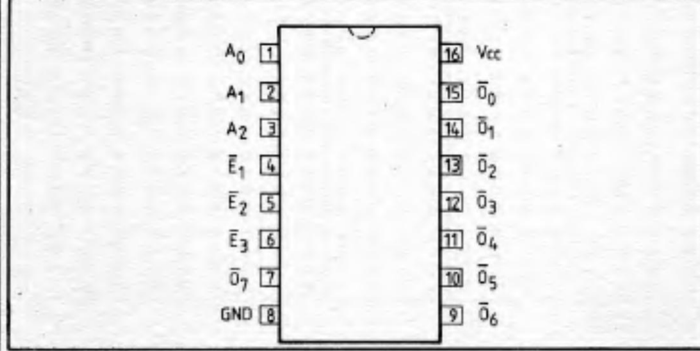


Fig. 3 : 74LS 138.

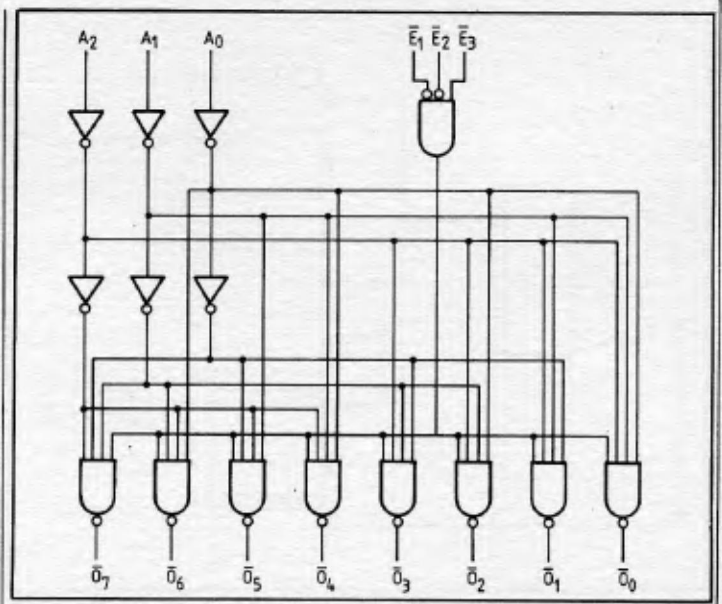


Schéma de principe du 74LS 138.

Plutôt qu'une description des broches, la table de vérité ci-dessous est plus parlante et évite un long discours.

TRUTH TABLE

INPUTS						OUTPUTS							
$\overline{E}_1$	$\overline{E}_2$	E_3	A_0	A_1	A_2	$\overline{O}_0$	$\overline{O}_1$	$\overline{O}_2$	$\overline{O}_3$	$\overline{O}_4$	$\overline{O}_5$	$\overline{O}_6$	$\overline{O}_7$
H	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	L	X	X	X	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	H	L	L	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	H	H	L	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	L	H	H
L	L	H	L	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

H = HIGH Voltage Level L = LOW Voltage Level X = Immaterial

Brochage du PIA 6821

Ce circuit est un périphérique d'entrée-sortie parallèle permettant sur le Micro Kit 09 de gérer toute la carte clavier (afficheur, clavier, etc...).
Le brochage de ce circuit est donné en figure 5.
- Vcc et Vss (broches 1 et 20) : lignes d'alimentation du 6821.

Vss = masse et Vcc = +5 V

La fonction de ce circuit dans le Micro Kit est de découper l'espace mémoire adressable par le 6809 en 8 blocs de 8 Koctets (8192 adresses) afin de fabriquer avec le bus d'adresses du 6809 les signaux CS des différents circuits présents sur la carte CPU.
Le découpage mémoire obtenu avec ce circuit est donné en figure 4.

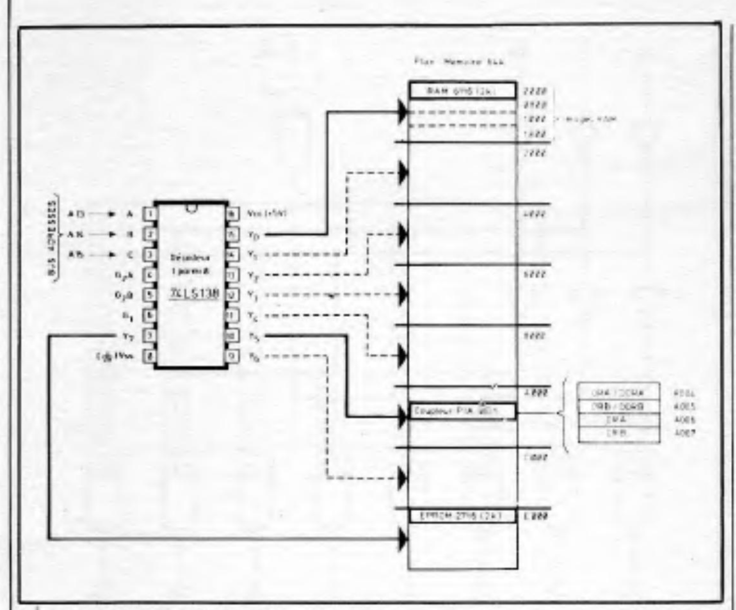


Fig. 4 : Découpage mémoire.

Fig. 5 : Le PIA 6821.

- PA0 à PA7 (broches 2 à 9) : Port A. Ce sont des lignes d'entrée/sortie programmables par le microprocesseur.
- PB0 à PB7 (broches 10 à 17) : Port B. Idem PA0 à PA7.
- CA2 et CA1 (broches 39 et 40) : lignes de dialogue du port A.
– CB1 et CB2 (broches 18 et 19) : lignes de dialogue du port B.
- D0 à D7 (broches 33 à 26) : bus de données du 6821 connecté directement au bus de données du CPU 6809.
- E (broche 25) : signal de synchronisation des échanges avec le 6809.
- R/$\overline{W}$ (broches 21) : signal de lecture-écriture venant du 6809.
- RS1 et RS0 (broches 35 et 36) ou Register Select : lignes de sélection des registres internes du circuit généralement connectées aux lignes A0 et A1 du microprocesseur.
- $\overline{RESET}$ (broche 34) : ligne d'initialisation du périphérique généralement connectée à la ligne $\overline{RESET}$ du CPU.
- $\overline{IRQA}$ et $\overline{IRQB}$ (broches 37 et 38) : lignes d'interruption du périphérique correspondant respectivement au port A et au port B.
- CS0, $\overline{CS2}$ et CS1 (broches 22 à 24) ou Chip Select : lignes de sélection du boîtier. CS0 et CS1 sont actives au niveau 1 logique alors que $\overline{CS2}$ est active au niveau bas.

Ainsi s'achève la description des circuits de la carte CPU. Nous allons maintenant pouvoir entrer dans le vif du sujet.

Carte périphérique (clavier)

Cette carte est composée du clavier évidemment, mais aussi des afficheurs, permettant de visualiser toutes les informations nécessaires, ainsi que les circuits de commande des afficheurs et l'interface cassette.

Description des circuits composant cette carte

- le 74LS240

Ce circuit est un octal Buffer couramment utilisé en microinformatique. Le brochage est donné figure 6.

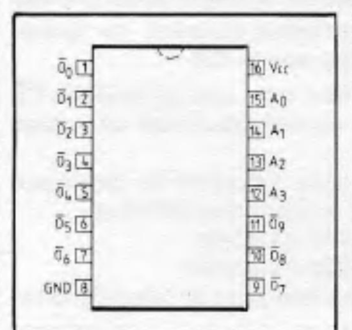


Fig. 6 : Le Buffer 74LS 240.

Fig. 7 : Brochage du 74 LS 42.

- Le 7442

C'est un multiplexeur démultiplexeur 1 parmi 10. Table de vérité et brochage en figure 7.

TRUTH TABLE

'42A • 'LS42 BCD INPUT				'43A EXCESS-3 INPUT				'44A EXCESS-3 GRAY INPUT				ALL TYPES DECIMAL OUTPUT									
A_3	A_2	A_1	A_0	A_3	A_2	A_1	A_0	A_3	A_2	A_1	A_0	$\overline{O_0}$	$\overline{O_1}$	$\overline{O_2}$	$\overline{O_3}$	$\overline{O_4}$	$\overline{O_5}$	$\overline{O_6}$	$\overline{O_7}$	$\overline{O_8}$	$\overline{O_9}$
L	L	L	L	L	L	H	H	L	L	H	L	L	H	H	H	H	H	H	H	H	H
L	L	L	H	L	H	L	L	L	H	H	L	H	L	H	H	H	H	H	H	H	H
L	L	H	L	L	H	L	H	L	H	H	H	H	H	L	H	H	H	H	H	H	H
L	L	H	H	L	H	H	L	L	H	L	H	H	H	H	L	H	H	H	H	H	H
L	H	L	L	L	H	H	H	L	H	L	L	H	H	H	H	L	H	H	H	H	H
L	H	L	H	H	L	L	L	H	H	L	L	H	H	H	H	H	L	H	H	H	H
L	H	H	L	H	L	L	H	H	H	L	H	H	H	H	H	H	H	L	H	H	H
L	H	H	H	H	L	H	L	H	H	H	H	H	H	H	H	H	H	H	L	H	H
H	L	L	L	H	L	H	H	H	H	H	L	H	H	H	H	H	H	H	H	L	H
H	L	L	H	H	H	L	L	H	L	H	L	H	H	H	H	H	H	H	H	H	L
H	L	H	L	H	H	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H
H	L	H	H	H	H	H	L	H	L	L	H	H	H	H	H	H	H	H	H	H	H
H	H	L	L	H	H	H	H	H	L	L	L	H	H	H	H	H	H	H	H	H	H
H	H	L	H	L	L	L	L	L	L	L	L	H	H	H	H	H	H	H	H	H	H
H	H	H	L	L	L	L	H	L	L	L	H	H	H	H	H	H	H	H	H	H	H
H	H	H	H	L	L	H	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H

H = HIGH Voltage Level
L = LOW Voltage Level

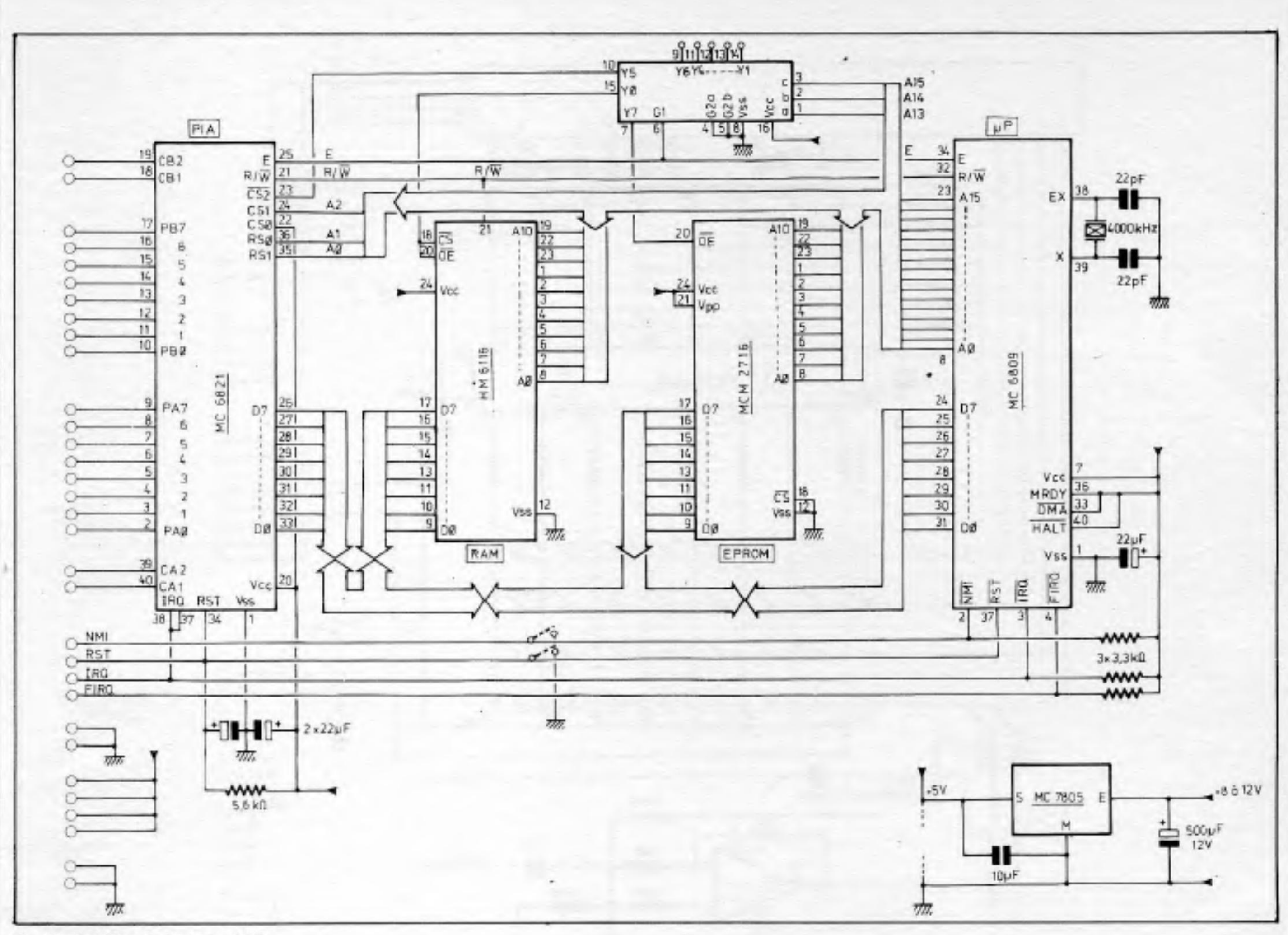


Fig. 8 : Schéma de la carte CPV.

MONTAGE DE LA MAQUETTE

Si le câblage des deux cartes du Micro Kit 09 ne présente pas vraiment de grosses difficultés, il n'en est pas de même pour la gravure des deux circuits imprimés en double face. Afin d'aider au maximum les lecteurs intéressés par cette réalisation, nous allons mettre à leur disposition le jeu de circuits, ceux-ci étant réalisés à trous métallisés afin de simplifier les interconnexions entre les deux faces.

Construction de la maquette

Voici quelques indications pour faciliter le montage de la maquette. Pour le montage des composants, s'aider des schémas des C.I. (fig. 11 à 13).

(1) Montage de la carte centrale (petite carte). Voir figures 8 et 13.

- Repérer la face composants (comportant l'inscription MICROKIT 09).
- Placer et souder les supports 40, 24 et 16 broches pour les circuits intégrés. Attention au sens (repère).
- Placer et souder les supports spéciaux d'interconnexion des cartes centrale et périphérique.
- Placer et souder résistances et condensateurs. Attention à la polarité des condensateurs électrolytiques.
- Placer et souder le quartz. (Ne pas trop enfoncer les pattes pour ne pas court-circuiter le boîtier avec une piste).
- Placer et souder les picots et fils d'alimentation ou le régulateur de tension et son radiateur.

(2) Montage de la carte périphérique (grande carte). Voir figures 9 et 12.

- Repérer la face touches (elle comporte l'inscription MICROKIT 09)
- Placer et souder les bases de touches.
- Préparer éventuellement les six supports 14 broches des afficheurs en coupant les pattes inutiles. Les placer et les souder.
- Couper et placer les barettes sécables à wrapper servant à l'interconnexion des cartes (2 × 14 et 2 × 16 broches). Attention à ne pas faire couler de la soudure sur les pattes à wrapper.
- Placer et souder l'inverseur de l'interface cassette.

Retourner la carte et câbler le dessous :

- Placer et souder les résistances
- Placer et souder les supports 20, 16 et 8 broches. Res-

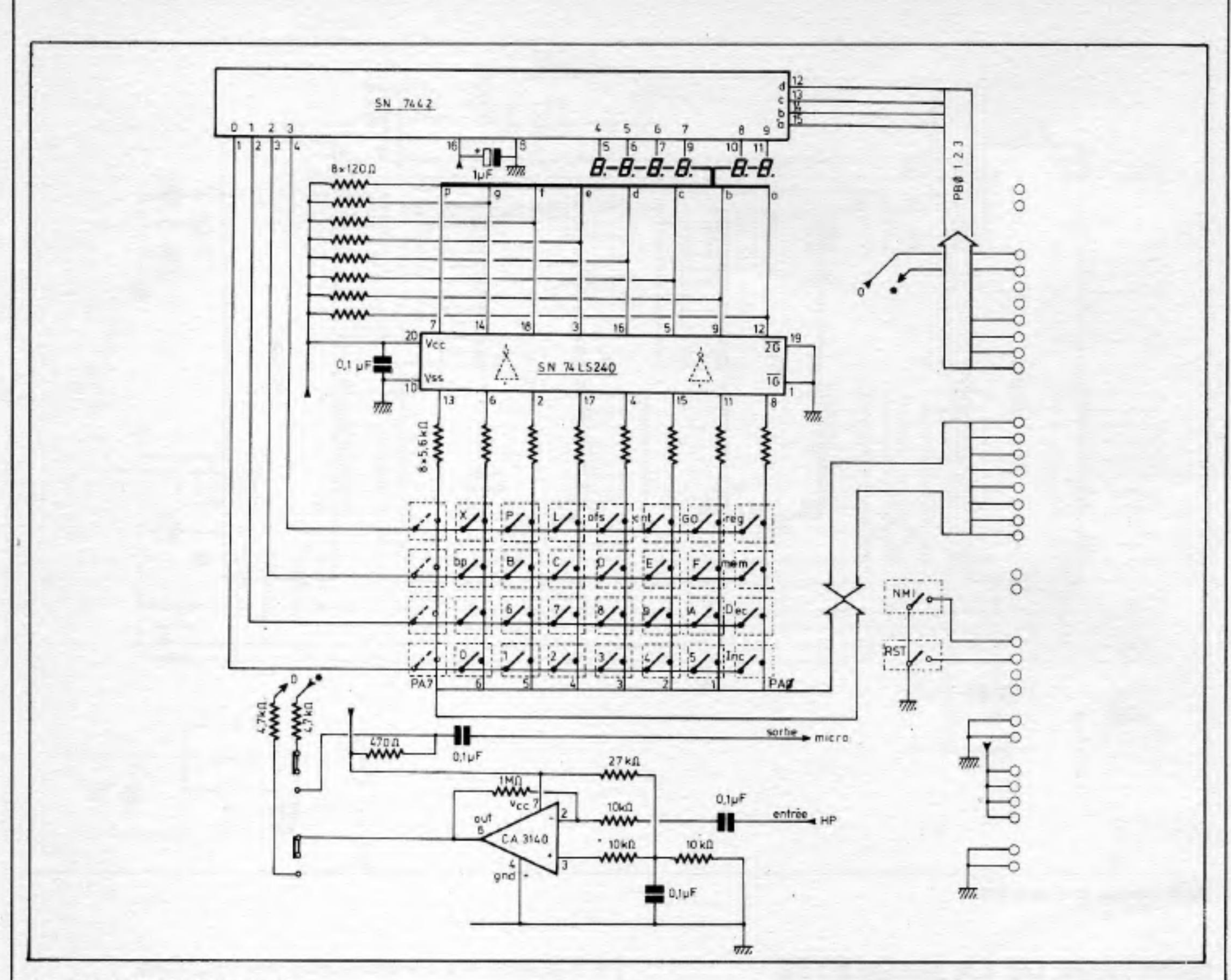


Fig. 9 : Schéma de la carte clavier.

pecter la position des repères.
- Placer et souder les condensateurs.

③ Vérifier soigneusement la valeur et la position de chaque composant, la bonne orientation des supports, la qualité de chaque soudure, l'absence de court-circuits dus à des gouttes de soudure.

④ Marquage des touches (fig. 10) : après un léger vernissage éventuel des capuchons des touches, en réaliser le marquage avec des lettres transfert, que l'on protégera par un nouveau léger vernissage.

⑤ Placer et souder les fils d'alimentation si l'on dispose d'une alimentation régulée + 5 V/500 mA. Sinon, placer et souder le régulateur intégré de tension avec ses capacités d'entrée-sortie et le connecter à l'adaptateur secteur délivrant 9-12 V.

En cas de panne secteur, on peut sauvegarder les données stockées en mémoire RAM dans le circuit 6116. Il suffit pour cela de couper la piste de circuit imprimé qui relie le circuit au + 5 V de l'alimentation et de la connecter à une pile de 4,5 V de sauvegarde. La piste est notée X, sur le dessin du dessous du circuit imprimé de la carte centrale en dernière page.

⑥ Insérer les circuits intégrés dans les supports. Attention au sens ! Voir repère 1 sur les figures 10 à 13 et vérifier.

⑦ Interconnecter les deux cartes, en n'oubliant pas les deux colonnettes intercartes avec ajustement éventuel par rondelles et en vérifiant le bon enfoncement des pattes à wrapper dans les supports DIL 16 broches de la carte centrale.

⑧ Brancher l'alimentation... Le symbole «-» doit apparaître sur le premier afficheur de gauche. Sinon, débrancher l'alimentation et vérifier...

Votre maquette semble fonctionner normalement. Nous allons dès maintenant apprendre à l'utiliser.

Afficheurs

Barrettes sécables
à wrapper

	X	P	L	ofs	cnt	GO	reg
	bpt	B	C	D	E	F	Mem
		6 u	7 pc	8	9	A	Dec
	Ø cc	1 a	2 b	3 dp	4 x	5 y	Inc

Inverseur
Interface K7

NMI

RST

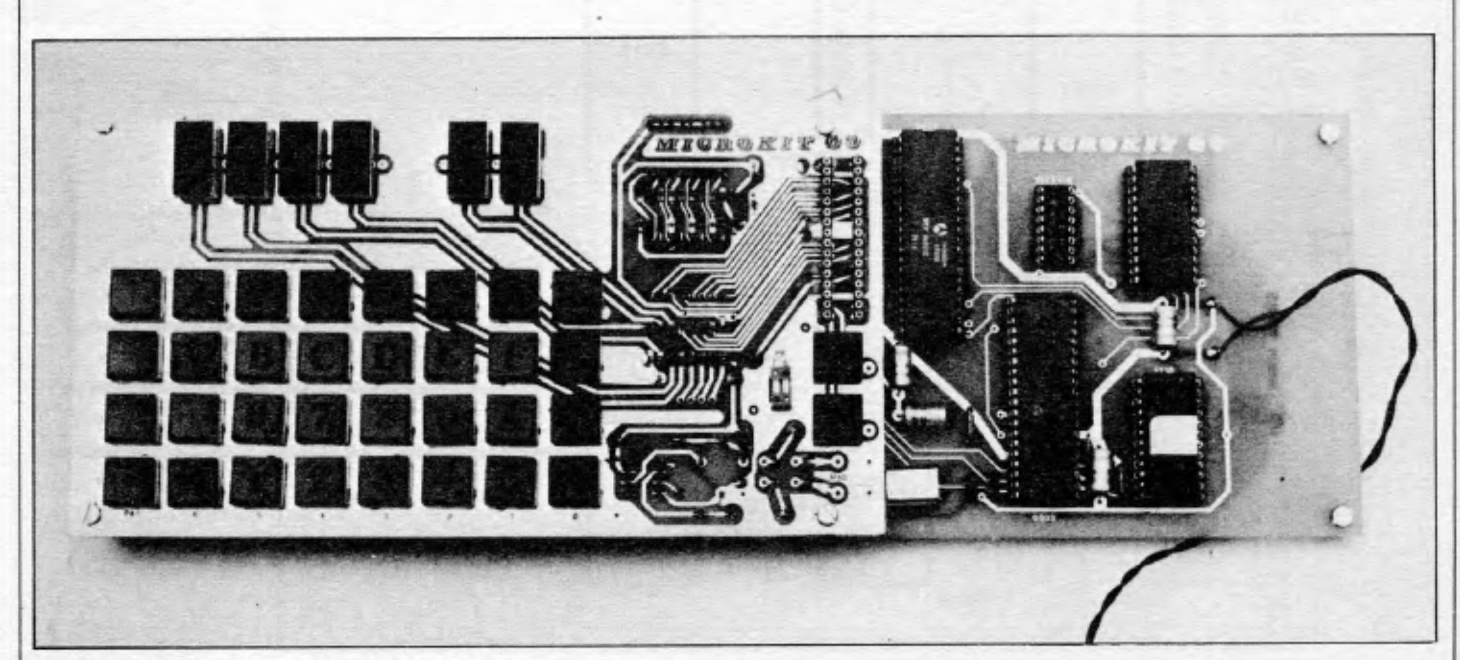
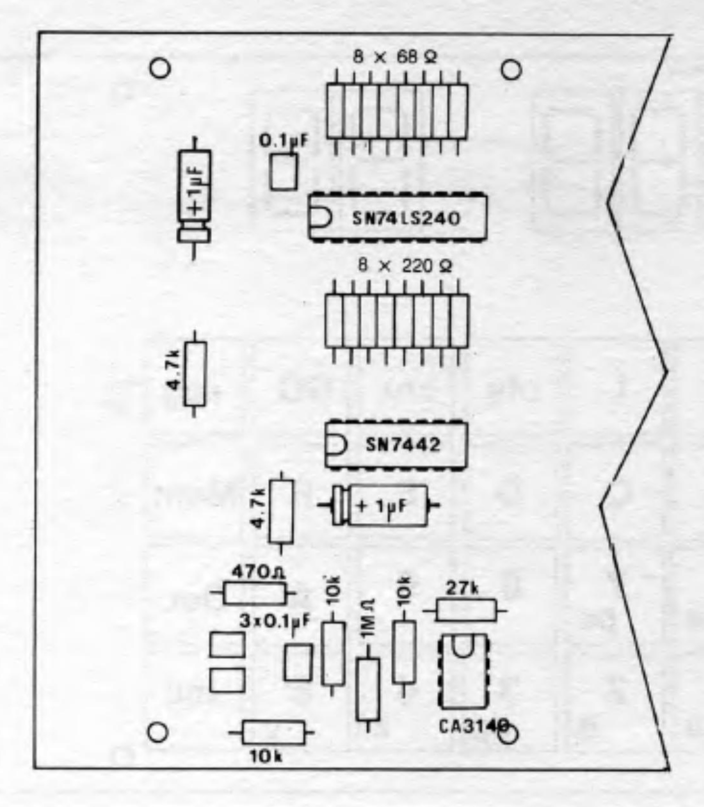


Fig. 10 : Organisation sérigraphique de la carte Clavier.



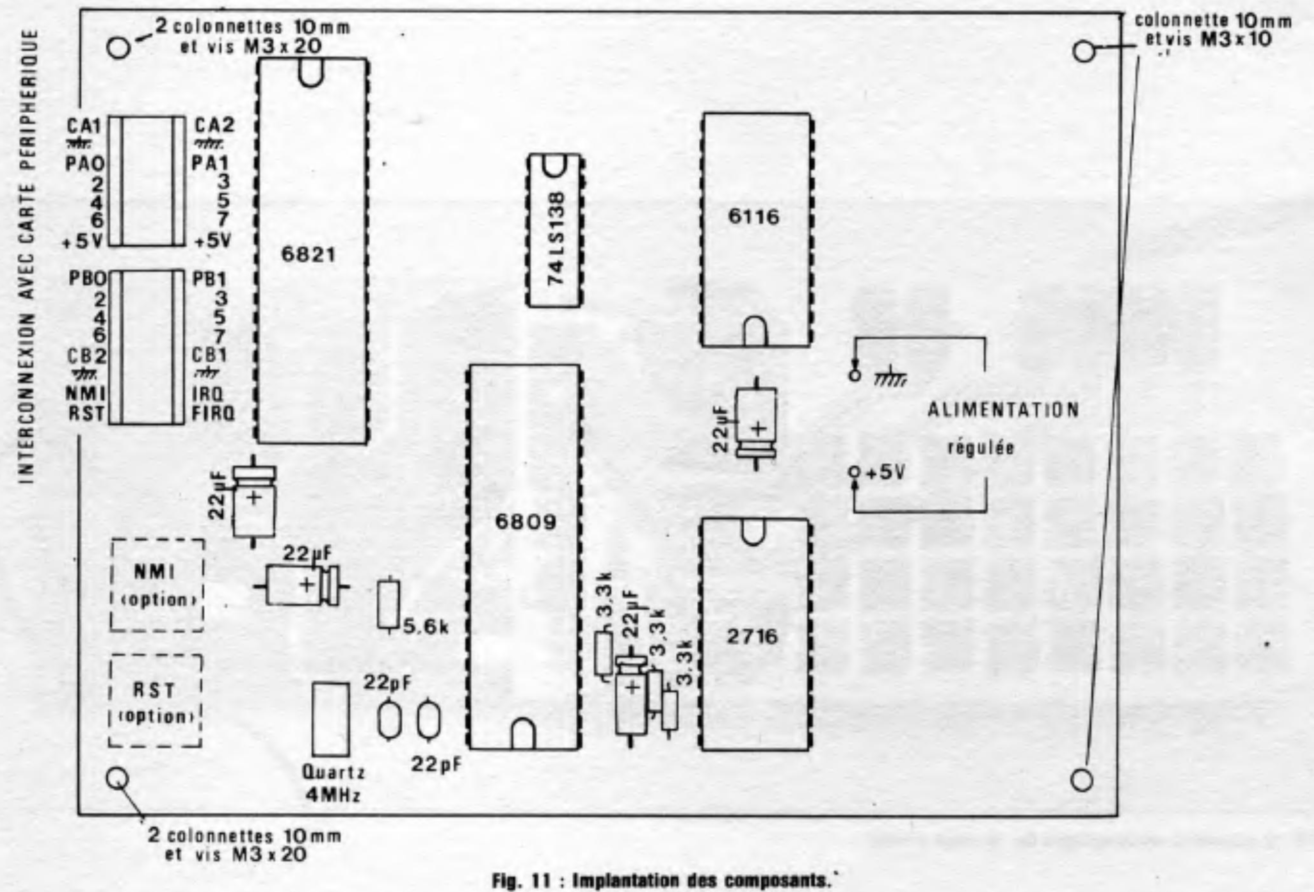


Fig. 11 : Implantation des composants.

MICROKIT Ø9

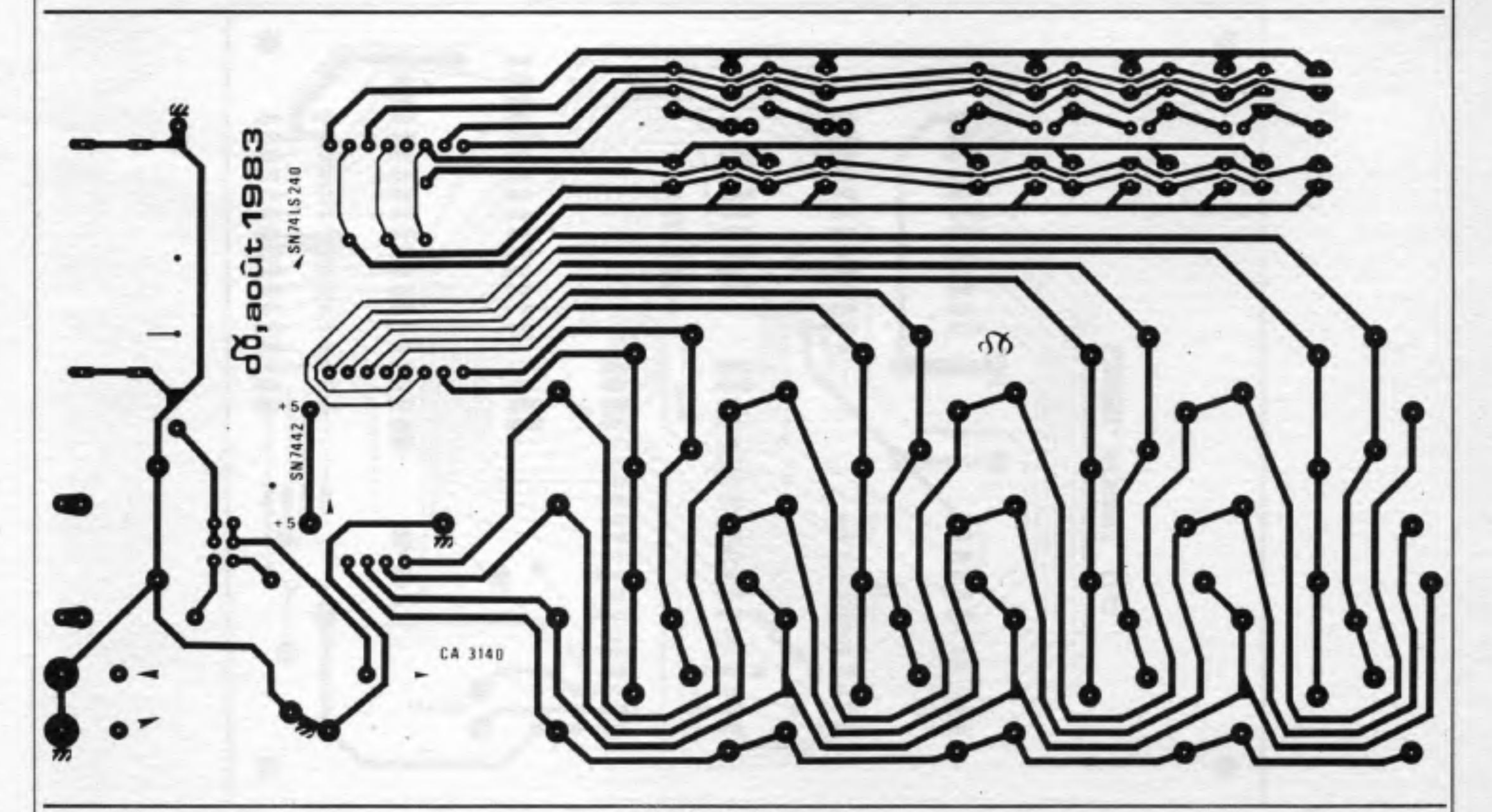
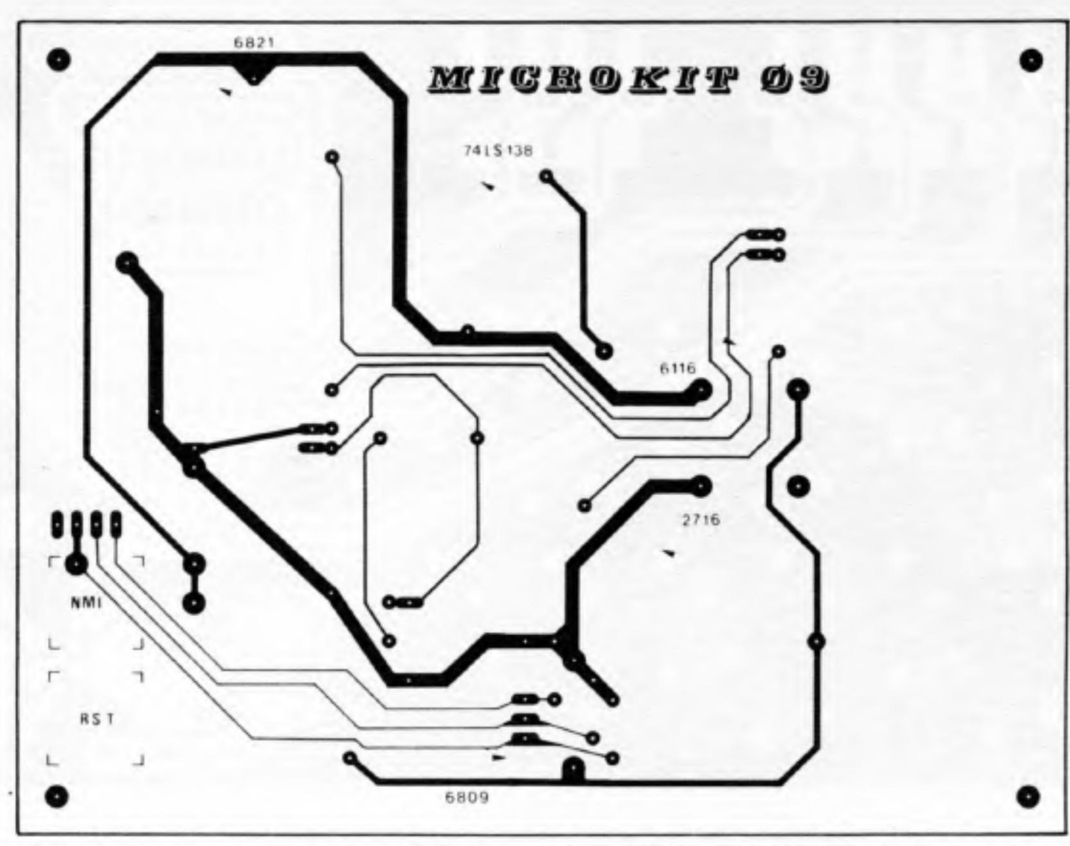


Fig. 12 : Carte Clavier.



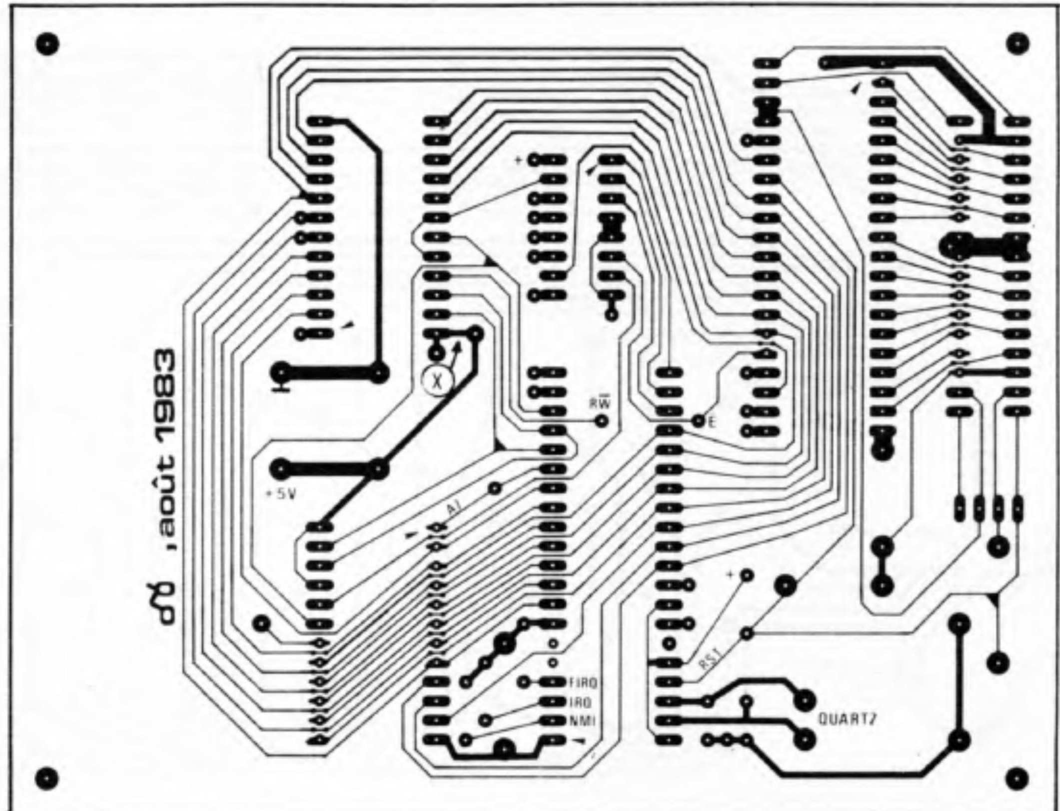


Fig. 13 : Carte Unité centrale.

L'ARCHITECTURE INTERNE DU 6809

L'architecture interne du 6809 est orientée 16 bits, ce qui fait de lui l'un des plus puissants microprocesseurs 8 bits du marché. En effet, ce microprocesseur possède deux accumulateurs 8 bits appelés A et B pouvant être associés pour former un seul registre 16 bits appelé D. Fig. 14.

Il possède également deux registres index X et Y (16 bits), deux registres pointeurs de piles U et S (16 bits) pouvant servir d'index, un pointeur de page DP (8 bits) permettant de découper l'espace mémoire adressable en 256 pages de 256 octets chacune et enfin un registre d'état de 8 bits CC. La figure ci-dessous donne le détail de l'architecture du 6809.

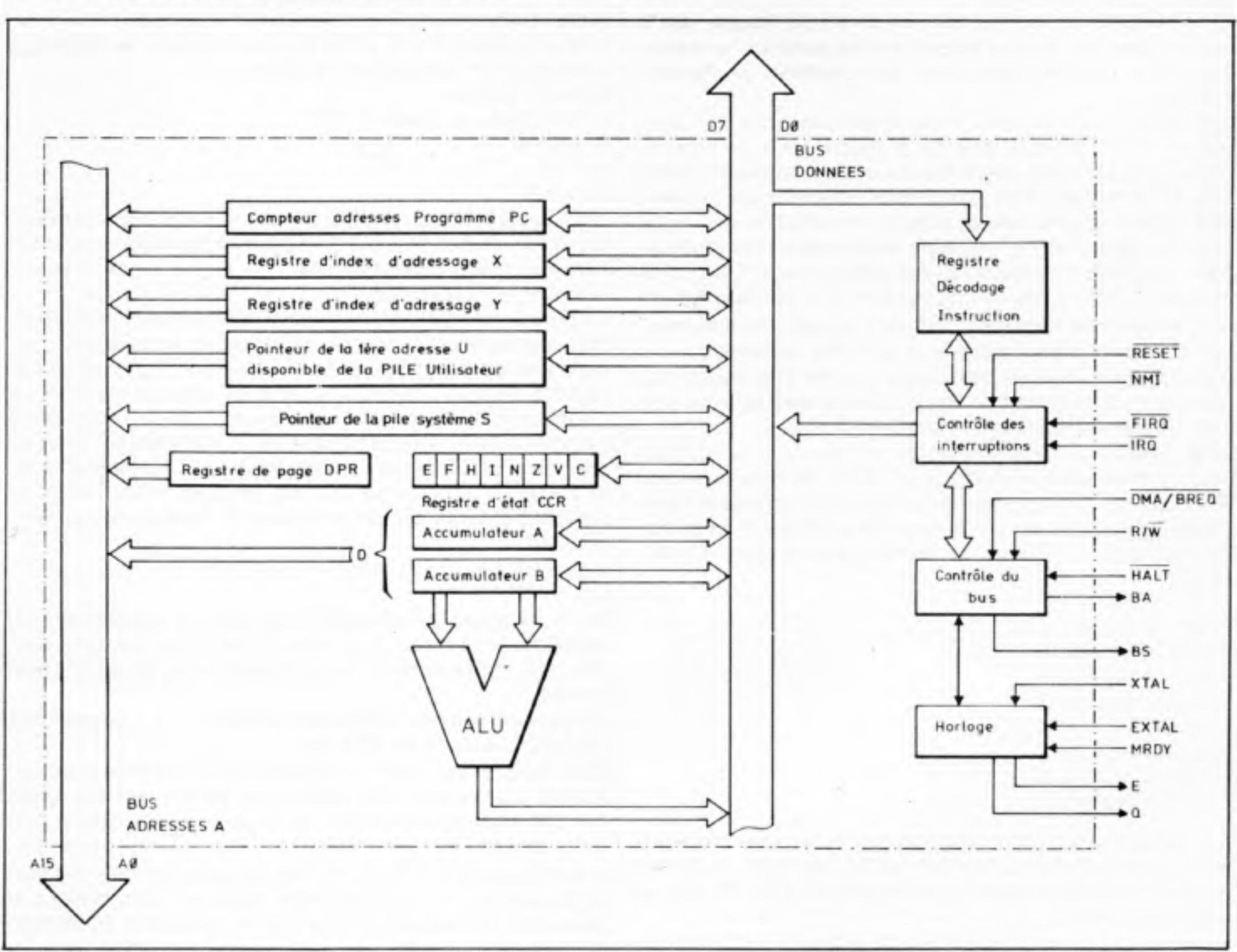


Fig. 14 : Architecture du 6809.

PROGRAMMATION - UTILISATION DE LA MAQUETTE

Maintenant que votre maquette fonctionne, il est temps de l'utiliser. Pour cela nous allons passer en revue toutes les touches de fonction du clavier.
Comme nous l'avons dit précédemment, à la mise sous tension, la barre centrale de l'afficheur de gauche doit s'allumer indiquant que le système est sous contrôle du moniteur et qu'il attend l'enfoncement d'une touche.

Touches de fonction

- Mem (mémoire) : cette touche permet la lecture et la modification de la case mémoire de votre choix. Dès l'appui sur cette touche, le symbole Π doit apparaître sur l'afficheur de droite.

Le numéro de la mémoire peut alors être entré et il se visualisera sur les quatre afficheurs de gauche.
Une fois l'adresse entrée, le contenu de cette adresse est visualisé sur les afficheurs de droite.
Dans le cas où vous voudriez modifier le contenu de cette

case mémoire, il suffit d'entrer la nouvelle donnée par l'appui successif sur deux des 16 touches hexadécimales. **Notons qu'un premier appui modifie le chiffre des poids faibles. Lors du second appui, ce chiffre est transféré à gauche sur l'afficheur des poids forts pour permettre d'entrer le chiffre des poids faibles. La nouvelle valeur entrée au clavier est mémorisée en RAM (si bien entendu, l'adresse spécifiée se trouve en RAM).**
Si par hasard, vous avez fait une erreur de frappe, continuez à taper les bonnes valeurs sur les touches hexadécimales. La donnée mémorisée sera toujours la dernière entrée.
Inc : touche «Increment». Permet d'incrémenter (= augmenter de 1 l'adresse lors de la visualisation du contenu d'une case mémoire. Cette touche est couramment utilisée lors de la rentrée d'un programme utilisateur en mémoire RAM. Cette touche permet aussi la visualisation des registres l'un après l'autre. Ce mode visualisation des registres peut soit être consécutif à une instruction d'interruption logicielle (SWI : code 3F) rencontrée lors du déroulement **d'un programme utilisateur, soit être appelé par l'introduction d'un point d'arrêt dans le programme utilisateur.**
Dec : touche «Décrement». Cette touche a le même rôle que la précédente mais au lieu d'incrémenter l'adresse d'un pas, elle la décrémente (= diminuer) d'un pas.
Reg : touche «Registre». Permet de visualiser le contenu des registres CCR, A, B, DP, X, Y, U, PC de l'unité centrale 6809 (fig. 4). Un appui sur la touche «reg» provoque l'affichage du symbole «Γ» sur le cinquième afficheur. La sélection du registre s'effectue alors grâce aux touches héxadécimales :

0 pour le registre CC
1 pour le registre A
2 pour le registre B
3 pour le registre DP
4 pour le registre X
5 pour le registre Y
6 pour le registre U
7 pour le registre PC

Le symbole du registre apparaît sur le sixième afficheur, alors que son contenu est affiché sur les deux ou quatre premiers afficheurs selon la configuration 8 ou 16 bits du registre examiné.

Adresse	Code instruction hexadécimal	Langage symbolique (mnémonique)	Commentaire
0100	12	NOP	Pas d'opération effectuée
0101	12	NOP	
0102	3F	SWI	Interruption du programme. Rappel du moniteur
0103	12	NOP	
0104	12	NOP	
0105	3F	SWI	

GO : touche de lancement d'un programme. Le programme se déroulera dès que l'adresse de départ aura été spécifiée à l'aide de l'appui sur quatre touches hexadécimales. Pour expliciter le rôle des différentes touches, dont le fonctionnement a déjà été présenté, nous conseillons le court programme en encadré ci-dessus.
Procédure à suivre :
RST → «–» sur le premier afficheur
Mem → «Π».
Entrer l'adresse 0100, n° de la case-mémoire de départ du programme → une donnée s'affiche.
Entrer 1 puis 2.
Inc → l'adresse passe à 0101.
Entrer 12.
Inc
Entrer 3F.
etc... jusqu'à l'adresse 0105. Lancer le programme par RST, GO 0100 ou par GO, GO 0100 si on effectuait auparavant l'examen d'une case-mémoire.
Résultat sur l'affichage
Le programme a été éxecuté jusqu'à l'instruction d'interruption logicielle SWI, ce qui provoque la visualisation des registres de l'unité centrale. Le premier visualisé est le registre d'état CC, symbolisé par **C**. En appuyant sur Inc, on visualise le contenu de A, de nouveau sur Inc celui de B puis DP, puis X, symbolisé par **H** (quatre afficheurs de gauche pour le contenu 16 bits de X) puis, Y puis U et enfin le compteur programme PC qui contient 0103, valeur de l'adresse pointée afin de poursuivre le déroulement du programme.
Appui sur Dex → on repasse au registre U.
Inc → PC
Inc → le symbole Γ apparaît. Il est dès lors possible soit de visualiser le contenu d'un registre particulier (revoir la touche reg), soit de revenir sous programme moniteur par appui sur Inc.
On peut s'entraîner à ces manipulations en recommençant l'opération à partir de GO0100.
Cnt : touche «continue». Elle permet de poursuivre le déroulement d'un programme, après que celui-ci eut été arrêté par une instruction SWI ou par un point d'arrêt. Relançons notre programme par RST GO 0100. Le compteur programme **pointe 0103. Un premier appui sur cnt fait apparaître le symbole «–». Un deuxième appui sur cnt provoque la poursuite du programme jusqu'à la deuxième instruction d'interruption codée 3F. On vérifie en effet que le compteur-programme pointe alors l'adresse 0106.**
bp : touche «break point» (point d'arrêt). Elle permet de placer un point d'arrêt dans un programme à l'adresse spécifiée par l'utilisateur.
Un premier appui sur bp provoque l'affichage de **bP** .
Entrer ensuite, par exemple, l'adresse 0102. Dès l'entrée du dernier chiffre d'adresse, le système passe sous con**trôle moniteur (symbole «–»). Relançons notre programme** GO 0100... Après exécution, le programme s'est arrêté en 0102 que pointe le PC. La poursuite s'effectue par cnt. L'utilisation du point d'arrêt permet la mise au point d'un programme («debuging») en vérifiant les registres de l'unité centrale.

ofs : touche «offset». Elle permet le calcul automatique de la valeur du déplacement lors de sauts ou de branchements relatifs dans un programme. Elle place la valeur hexadécimale de ce déplacement dans le programme utilisateur, situé en RAM. Ecrivons le petit programme, en encadré ci-dessous, afin de s'exercer à l'utilisation de cette touche : On écrit ce programme à partir de l'adresse 0200. On y entre B6, puis 00, ou encore 00 et enfin 27 à l'adresse 0203. L'instruction de branchement BEQ «branch if equal») comprend le code opération 27 qui doit être suivi de la valeur du déplacement exprimé en hexa (en code complément à 2/). Le calcul de cette valeur est sur le Microkit 09 automatique. Sans incrémenter (on est toujours à l'adresse 0203, donnée 27) appuyer sur ofs → donnera le symbole « A r ». Entrer par le clavier l'adresse de destination du branchement 0220 puis GO → . Le système place alors la valeur 1B (valeur en code complément à 2 du déplacement), à l'adresse 0204. Appuyer sur Inc, et entrer 3F. Puis entrer 3F à l'adresse 0220. Lancer le programme GO 0200. Après l'exécution du programme, le PC pointe l'adresse 0206 ou 0221 selon le contenu de la case-mémoire 0000, contenu que l'on retrouve d'ailleurs dans A.

P : touche «Punch» = enregistrement. Cette touche permet de transférer une zone RAM vers un magnétophone à cassette, pour y stocker programmes et données. Un appui sur P entraîne l'apparition du symbole S (start). Entrer l'adresse de début de la zone à transférer. Dès l'entrée du quatrième chiffre de l'adresse, le symbole F (fin) apparaît. Il faut entrer l'adresse de fin de la zone mémoire à transférer. Dès l'entrée du quatrième chiffre de l'adresse de fin, la transmission vers la cassette s'effectue. Le temps de transmission sera fonction de la longueur du bloc à transmettre. En fin de transmission, le symbole F apparaît sur le premier afficheur. Exercices : calculer la vitesse approximative de transmission... sec/2 koctets).

L : touche «Load» = lecture. Elle permet de charger en mémoire RAM un programme provenant d'un magnétophone à cassette.

X : cette touche combinée avec ofs permet le calcul automatique de déplacement en adressage indexé.

LES INSTRUCTIONS DU 6809

Le 6809 possède un grand jeu d'instructions donné dans le tableau 1, lequel nous est fourni par le fabricant (MOTOROLA). Toutes ces instructions peuvent être classées en plusieurs catégories distinctes.

Les instructions arithmétiques

ADD : addition du contenu d'une case mémoire avec un accumulateur
ADC : addition du contenu d'une case mémoire avec un accumulateur plus la retenue
ABX : addition du registre B avec le registre X
DAA : ajustement décimal de l'accumulateur A
MUL : multiplication de l'accumulateur A par l'accumulateur B (non signée)
SUB : soustraction du contenu d'une case mémoire à l'accumulateur
SBC : soustraction du contenu d'une case mémoire à l'accumulateur moins la retenue
SEX : extension de signes de l'accumulateur B à l'accumulateur A.

Les instructions de rotation et de décalage

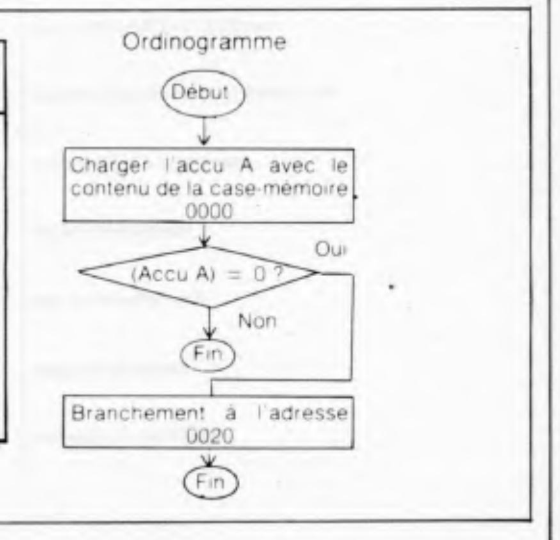
ASR : décalage arithmétique à droite
LSL ou ASL : décalage logique ou arithmétique à gauche
LSR : décalage logique à droite
ROL : rotation à gauche
ROR : rotation à droite
Ces instructions fonctionnent avec les accumulateurs A et B ainsi qu'avec n'importe quelle case mémoire.

Les intructions logiques

AND : «ET logique» entre une case mémoire et un registre interne (A, B, CC).
EOR : «OU exclusif» entre une case mémoire et un registre interne (A, B, CC).
OR : «OU logique» entre une case mémoire et un registre interne (A, B, CC).

Etiquette	Adresse	Code instruction Opération	Code instruction Opérande	Mnémonique
	0200	B6	00 00	LDA > 0000
	0203	27	□ □ Valeur hexa du déplacement	BEQ END
déplacement ↓	0205	3F		SWI
END	0220	3F		SWI

Programme permettant de s'exercer à l'utilisation de la touche Ofs.



OPERATION	Mnémonique Instruction	Modes d'adressage du 6809 INHERENT OP	~	N	DIRECT < OP	~	N	EXTENDED > OP	~	N	IMMEDIAT # OP	~	N	INDEXE ① OP	~	N	RELATIF ⑤ OP	~	N	DESCRIPTION	Indicateurs 5 H	3 N	2 Z	1 V	0 C
Addition de l'accumulateur B à X (non signé)	ABX	3A	3	1																B + X → X	•	•	•	•	•
Addition du contenu mémoire à l'accumulateur, avec retenue	ADCA				99	4	2	B9	5	3	89	2	2	A9	4+	2+				A+M+C→A	↕	↕	↕	↕	↕
	ADCB				D9	4	2	F9	5	3	C9	2	2	E9	4+	2+				B+M+C→B	↕	↕	↕	↕	↕
Addition du contenu mémoire à l'accumulateur	ADDA				9B	4	2	BB	5	3	8B	2	2	AB	4+	2+				A+M→A	↕	↕	↕	↕	↕
	ADDB				DB	4	2	FB	5	3	CB	2	2	EB	4+	2+				B+M→B	↕	↕	↕	↕	↕
	ADDD				D3	6	2	F3	7	3	C3	4	3	E3	6+	2+				D+M:M+1→D	↕	↕	↕	↕	↕
ET logique entre mémoire et l'accumulateur	ANDA				94	4	2	B4	5	3	84	2	2	A4	4+	2+				A . M → A	•	↕	↕	0	•
	ANDB				D4	4	2	F4	5	3	C4	2	2	E4	4+	2+				B . M → B	•	↕	↕	0	•
« ET » logique avec le registre codes condition	ANDCC										1C	3	2							CC.IMM→CC					1
Décalage arithmétique à gauche du conteu mémoire ou accumulateur	ASLA	48	2	1																A	8	↕	↕	↕	↕
	ASLB	58	2	1																B	8	↕	↕	↕	↕
	ASL				08	6	2	78	7	3				68	6+	2+				M	8	↕	↕	↕	↕
Décalage arithmétique à droite du contenu mémoire ou accumulateur	ASRA	47	2	1																A	8	↕	↕	•	↕
	ASRB	57	2	1																B	8	↕	↕	•	↕
	ASR				07	6	2	77	7	3				67	6+	2–				M	8	↕	↕	•	↕
Branchement si pas de retenue	BCC																24	3	2	Si C = 0	•	•	•	•	•
	LBCC																10 24	5(6)	4		•	•	•	•	•
Branchement si retenue	BCS																25	3	2	Si C = 1	•	•	•	•	•
	LBCS																10 25	5(6)	4		•	•	•	•	•
Branchement si égal	BEQ																27	3	2	Si z = 1	•	•	•	•	•
	LBEQ																10 27	5(6)	4		•	•	•	•	•
Branchement si supérieur ou égal (signé)	BGE																2C	3	2	Si ≥ Zéro	•	•	•	•	•
	LBGE																10 2C	5(6)	4		•	•	•	•	•
Branchement si supérieur (signé)	BGT																2E	3	2	Si > Zéro	•	•	•	•	•
	LBGT																10 2E	5(6)	4		•	•	•	•	•
Branchement si supérieur (non signé)	BHI																22	3	2		•	•	•	•	•
	LBHI																10 22	5(6)	4		•	•	•	•	•
Branchement si supérieur ou égal (non signé)	BHS																24	3	2		•	•	•	•	•
	LBHS																10 24	5(6)	4		•	•	•	•	•
Test de bit mémoire avec l'accumulateur	BITA				95	4	2	B5	5	3	85	2	2	A5	4+	2+				(M ∧ A)	•	↕	↕	0	•
	BITB				D5	4	2	F5	5	3	C5	2	2	E5	4+	2+				(M ∧ B)	•	↕	↕	0	•
Branchement si inférieur ou égal (signé)	BLE																2F	3	2	Si ≤ Zéro	•	•	•	•	•
	LBLE																10 2F	5(6)	4		•	•	•	•	•
Branchement si inférieur (non signé)	BLO																25	3	2		•	•	•	•	•
	LBLO																10 25	5(6)	4		•	•	•	•	•
Branchement si inférieur ou égal (non signé)	BLS																23	3	2		•	•	•	•	•
	LBLS																10 23	5(6)	4		•	•	•	•	•
Branchement si inférieur (signé)	BLT																2D	3	2	Si < Zéro	•	•	•	•	•
	LBLT																10 2D	5(6)	4		•	•	•	•	•
Branchement si négatif	BMI																2B	3	2		•	•	•	•	•
	LBMI																10 2B	5(6)	4		•	•	•	•	•
Branchement si non égal	BNE																26	3	2	Si Z = 0	•	•	•	•	•
	LBNE																10 26	5(6)	4		•	•	•	•	•
Branchement si positif	BPL																2A	3	2		•	•	•	•	•
	LBPL																10 2A	5(6)	4		•	•	•	•	•
Branchement inconditionnel	BRA																20	3	2		•	•	•	•	•
	LBRA																16	5	3		•	•	•	•	•

Tableau 1

CODOP — nb de top d'horloge — nb d'octets

OPÉRATION	Mnémonique Instruction	INHERENT			DIRECT			EXTENDED			IMMEDIAT			INDEXE			RELATIF			DESCRIPTION	Indicateurs H	N	Z	V	C
		OP	~	N	OP	~	N	OP	~	N	OP	~	N	OP	~	N	OP	~	N						
Non branchement	BRN																21	3	2		•	•	•	•	•
	LBRN																10 21	5	4		•	•	•	•	•
Branchement à un sous-programme	BSR																8D	7	2		•	•	•	•	•
	LBSR																17	9	3		•	•	•	•	•
Branchement si pas de débordement = 0	BVC																28	3	2	Si V = 0	•	•	•	•	•
	LBVC																10 28	5(6)	4		•	•	•	•	•
Branchement si débordement = 1	BVS																29	3	2	Si V = 1	•	•	•	•	•
	LBVS																10 29	5(6)	4		•	•	•	•	•
Mise à zéro du contenu de l'accumulateur ou de la mémoire	CLRA	4F	2	1																0 → A	•	0	1	0	0
	CLRB	5F	2	1																0 → B	•	0	1	0	0
	CLR				0F	6	2	7F	7	3				6F	6+	2+				0 → M	•	0	1	0	0
Comparaison du contenu mémoire avec l'accumulateur	CMPA				91	4	2	B1	5	3	81	2	2	A1	4+	2+				M avec A	⑧	↕	↕	↕	↕
	CMPB				D1	4	2	F1	5	3	C1	2	2	E1	4+	2+				M avec B	⑧	↕	↕	↕	↕
	CMPD				10 93	7	3	10 B3	8	4	10 83	5	4	10 A3	7+	3+				M:M + 1 avec D	•	↕	↕	↕	↕
Comparaison mémoire avec pointeur de pile	CMPS				11 9C	7	3	11 BC	8	4	11 8C	5	4	11 AC	7+	3+				M:M + 1 avec S	•	↕	↕	↕	↕
	CMPU				11 93	7	3	11 B3	8	4	11 83	5	4	11 A3	7+	3+				M:M + 1 avec U	•	↕	↕	↕	↕
Comparaison mémoire avec registre index	CMPX				9C	6	2	BC	7	3	8C	4	3	AC	6+	2+				M:M + 1 avec X	•	↕	↕	↕	↕
	CMPY				10 9C	7	3	10 BC	8	4	10 8C	5	4	10 AC	7+	3+				M:M + 1 avec Y	•	↕	↕	↕	↕
Complément à deux de l'accumulateur ou du contenu mémoire	COMA	43	2	1																$\overline{A}$ → A	•	↕	↕	0	1
	COMB	53	2	1																$\overline{B}$ → B	•	↕	↕	0	1
	COM				03	6	2	73	7	3				63	6+	2+				$\overline{M}$ → M	•	↕	↕	0	1
« ET » logique avec le registre codes condition puis attente d'interruption	CWAI	3C	20	2																CC . XX → CC					1
Ajustement décimal de l'accumulateur A	DAA	19	2	1																	•	↕	↕	0	↕
Décrémentation du contenu mémoire ou de l'accumulateur	DECA	4A	2	1																A − 1 → A	•	↕	↕	↕	•
	DECB	5A	2	1																B − 1 → B	•	↕	↕	↕	•
	DEC				0A	6	2	7A	7	3				6A	6+	2+				M − 1 → M	•	↕	↕	↕	•
« OU » exclusif du contenu mémoire avec l'accumulateur	EORA				98	4	2	B8	5	3	88	2	2	A8	4+	2+				A M → A	•	↕	↕	0	•
	EORB				D8	4	2	F8	5	3	C8	2	2	E8	4+	2+				B M → B	•	↕	↕	0	•
Echange de R_1 avec R_2 (R_1, R_2 = A, B, CC, DP)	EXG R1, R2	1E	7	2																R1 → R2 ②	•	•	•	•	•
Incrémentation du contenu mémoire ou de l'accumulateur	INCA	4C	2	1																A + 1 → A	•	↕	↕	↕	•
	INCB	5C	2	1																B + 1 → B	•	↕	↕	↕	•
	INC				0C	6	2	7C	7	3				6C	6+	2+				M + 1 → M ③	•	↕	↕	↕	•
Saut inconditionnel	JMP				0E	3	2	7E	4	3				6E	3+	2+				EA → PC	•	•	•	•	•
Saut à un sous-programme	JSR				9D	7	2	BD	8	3				AD	7+	2+					•	•	•	•	•
Chargement de l'accumulateur avec le contenu mémoire	LDA				96	4	2	B6	5	3	86	2	2	A6	4+	2+				M → A	•	↕	↕	0	•
	LDB				D6	4	2	F6	5	3	C6	2	2	E6	4+	2+				M → B	•	↕	↕	0	•
	LDD				DC	5	2	FC	6	3	CC	3	3	EC	5+	2+				M:M + 1 → D	•	↕	↕	0	•
Chargement du pointeur de pile avec le contenu mémoire	LDS				10 DE	6	3	10 FE	7	4	10 CE	4	4	10 EE	6+	3+				M:M + 1 → S	•	↕	↕	0	•
	LDU				DE	5	2	FE	6	3	CE	3	3	EE	5+	2+				M:M + 1 → U	•	↕	↕	0	•
Chargement du registre index avec le contenu mémoire	LDX				9E	5	2	BE	6	3	8E	3	3	AE	5+	2+				M:M + 1 → X	•	↕	↕	0	•
	LDY				10 9E	6	3	10 BE	7	4	10 8E	4	4	10 AE	6+	3+				M:M + 1 → Y	•	↕	↕	0	•
Chargement de l'adresse effective dans le pointeur de pile	LEAS													32	4+	2+				EA → S ③	•	•	•	•	•
	LEAU													33	4+	2+				EA → U	•	•	•	•	•
Chargement de l'adresse effective dans le registre index	LEAX													30	4+	2+				EA → X	•	•	↕	•	•
	LEAY													31	4+	2+				EA → Y	•	•	↕	•	•
Décalage logique à gauche du contenu mémoire ou de l'accumulateur	LSLA	48	2	1																A	•	↕	↕	↕	↕
	LSLB	58	2	1																B	•	↕	↕	↕	↕
	LSL				08	6	2	78	7	3				68	6+	2+				M	•	↕	↕	↕	↕
Décalage logique à droite du contenu mémoire ou de l'accumulateur	LSRA	44	2	1																A	•	0	↕	•	↕
	LSRB	54	2	1																B	•	0	↕	•	↕
	LSR				04	6	2	74	7	3				64	6+	2+				M	•	0	↕	•	↕
Multiplication non signée	MUL	3D	11	1																A × B → D	•	•	↕	•	⑨
Complément à un du contenu mémoire ou de l'accumulateur	NEGA	40	2	1																$\overline{A}$ + 1 → A	8	↕	↕	↕	↕
	NEGB	50	2	1																$\overline{B}$ + 1 → B	8	↕	↕	↕	↕
	NEG				00	6	2	70	7	3				60	6+	2+				$\overline{M}$ + 1 → M	8	↕	↕	↕	↕

Tableau 1 (suite)

OPÉRATION	Mnémonique Instruction	INHERENT OP	INHERENT ~	INHERENT N	DIRECT OP	DIRECT ~	DIRECT N	EXTENDED OP	EXTENDED ~	EXTENDED N	IMMEDIAT OP	IMMEDIAT ~ #	IMMEDIAT N	INDEXE OP	INDEXE ~ 1	INDEXE N	RELATIF OP	RELATIF ~ 5	RELATIF N	DESCRIPTION	5 H	3 N	2 Z	1 V	0 C
Non opération	NOP	12	2	1																	•	•	•	•	•
« OU » logique mémoire et accumulateur	ORA				9A	4	2	BA	5	3	8A	2	2	AA	4 +	2 +				A M→A	•	I	I	0	•
	ORB				DA	4	2	FA	5	3	CA	2	2	EA	4 +	2 +				B M→B	•	I	I	0	•
« OU » logique avec le registre codes condition	ORCC										1A	3	2							CC IMM→CC					7
Empilement de tout(s) registre(s) (sauf S) sur la pile S	PSHS	34	5 + (4)	2																	•	•	•	•	•
Empilement de tout(s) registre(s) (sauf U) sur la pile U	PSHU	36	5 +	2																	•	•	•	•	•
Dépilement de tout(s) registre(s) de la pile S	PULS	35	5 +	2																	•	•	•	•	•
Dépilement de tout(s) registre(s) (sauf U) de la pile U	PULU	37	5 +	2																	•	•	•	•	•
	ROLA	49	2	1																A	•	I	I	I	I
Décalage circulaire à gauche du contenu mémoire ou de l'accumulateur	ROLB	59	2	1																B	•	I	I	I	I
	ROL				09	6	2	79	7	3				69	6 +	2 +				M	•	I	I	I	I
	RORA	46	2	1																A	•	I	I	•	I
Décalage circulaire à droite du contenu mémoire ou de l'accumulateur	RORB	56	2	1																B	•	I	I	•	I
	ROR				06	6	2	76	7	3				66	6 +	2 +				M	•	I	I	•	I
Retour d'interruption	RTI	3B	6/15	1																					(7)
Retour de sous-programme	RTS	39	5	1																	•	•	•	•	•
Soustraction du contenu mémoire de l'accumulateur	SBCA				92	4	2	B2	5	3	82	2	2	A2	4 +	2 +				A − M − C→A	(8)	I	I	I	I
	SBCB				D2	4	2	F2	5	3	C2	2	2	E2	4 +	2 +				B − M − C→B	(8)	I	I	I	I
Extension de signe de l'accumulateur B à l'accumulateur A	SEX	1D	2	1																	•	I	I	0	•
	STA				97	4	2	B7	5	3				A7	4 +	2 +				A→M	•	I	I	0	•
Mise en mémoire du contenu de l'accumulateur	STB				D7	4	2	F7	5	3				E7	4 +	2 +				B→M	•	I	I	0	•
	STD				DD	5	2	FD	6	3				ED	5 +	2 +				D→M:M + 1	•	I	I	0	•
Mise en mémoire du pointeur de pile	STS				10 DF	6	3	10 FF	7	4				10 EF	6 +	3 +				S→M:M + 1	•	I	I	0	•
	STU				DF	5	2	FF	6	3				EF	5 +	2 +				U→M:M + 1	•	I	I	0	•
Mise en mémoire du registre index	STX				9F	5	2	BF	6	3				AF	5 +	2 +				X→M:M + 1	•	I	I	0	•
	STY				10 9F	6	3	10 BF	7	4				10 AF	6 +	3 +				Y→M:M + 1	•	I	I	0	•
	SUBA				90	4	2	B0	5	3	80	2	2	A0	4 −	2 −				A − M→A	(8)	I	I	I	I
Soustraction du contenu mémoire de l'accumulateur	SUBB				D0	4	2	F0	5	3	C0	2	2	E0	4 +	2 +				B − M→B	(8)	I	I	I	I
	SUBD				93	6	2	B3	7	3	83	4	3	A3	6 +	2 +				D − M:M + 1→D	•	I	I	I	I
Interruption programmée	SW1 (6)	3F	19	1																	•	•	•	•	•
	SW12	10 3F	20	2																	•	•	•	•	•
	SW13	11 3F	20	2																	•	•	•	•	•
Synchronisation avec la ligne d'interruption	SYNC	≥13	2	1																	•	•	•	•	•
Transfert de R1 à R2	TFR R1, R2	1F	7	2																R1→R2 (2)	•	•	•	•	•
	TSTA	4D	2	1																	•	I	I	0	•
Test mémoire ou accumulateur	TSTB	5D	2	1																	•	I	I	0	•
	TST				0D	6	2	7D	7	3				6D	6 +	2 +					•	I	I	0	•

Tableau I (suite)

Légende

OP	code hexadécimal de l'opération	Z	Zéro (octet)
~	nombre de cycles - horloge	V	Dépassement
N	nombre d'octets	C	report du bit 7
+	plus arithmétique	I	testé et mis à 1 si condition vraie, sinon mis à zéro
−	moins arithmétique	•	non affecté
×	multiplication	CC	registre d'indicateur d'état
$\overline{M}$	complément de M	:	concatenation
→	transfert dans	V	OU logique
H	demi-retenue du bit 3	∀	OU logique Ex
N	négatif (bit de signe)	Λ	ET logique

SOURCE	DESTINATION
0000 = D (A B)	1000 = A
0001 = X	1001 = B
0010 = Y	1010 = CCR
0011 = U	1011 = DPR
0100 = S	
0101 = PC	

Code registre

Notes :
1. Dans le tableau sont donnés le nombre de cycles et d'octets de base. Pour déterminer le nombre total de cycles et d'octets ajouter les valeurs du tableau 2 des types d'adressage indexé.
2. R1 et R2 peuvent être une paire de registres 8 bits : A, B, CC, DP, ou 16 bits : X, Y, U, S, D, PC.
Il faut alors ajouter au code-instruction un post-octet selon le code-registre

Tableau 1 (suite)

Type	Forms	Non Indirect				Indirect			
		Assembler Form	Postbyte OP Code	+ ~	+ #	Assembler Form	Postbyte OP Code	+ ~	+ #
Constant Offset From R (2's Complement Offsets)	No Offset	,R	1RR00100	0	0	[,R]	1RR10100	3	0
	5 Bit Offset	n, R	0RRnnnnn	1	0	defaults to 8-bit			
	8 Bit Offset	n, R	1RR01000	1	1	[n, R]	1RR11000	4	1
	16 Bit Offset	n, R	1RR01001	4	2	[n, R]	1RR11001	7	2
Accumulator Offset From R (2's Complement Offsets)	A Register Offset	A, R	1RR00110	1	0	[A, R]	1RR10110	4	0
	B Register Offset	B, R	1RR00101	1	0	[B, R]	1RR10101	4	0
	D Register Offset	D, R	1RR01011	4	0	[D, R]	1RR11011	7	0
Auto Increment/Decrement R	Increment By 1	,R +	1RR00000	2	0	not allowed			
	Increment By 2	,R + +	1RR00001	3	0	[,R + +]	1RR10001	6	0
	Decrement By 1	,– R	1RR00010	2	0	not allowed			
	Decrement By 2	,– – R	1RR00011	3	0	[,– – R]	1RR10011	6	0
Constant Offset From PC (2's Complement Offsets)	8 Bit Offset	n, PCR	1xx01100	1	1	[n, PCR]	1xx11100	4	1
	16 Bit Offset	n, PCR	1xx01101	5	2	[n, PCR]	1xx11101	8	2
Extended Indirect	16 Bit Address	–	–	–	–	[n]	10011111	5	2

R = X, Y, U or S
x = Don't Care

RR
00 = X
01 = Y
10 = U
11 = S

Tableau I (suite et fin)

Les instructions d'incrémentation/décrémentation, mise à zéro, complémentation.

CLR : mise à 0 d'un accumulateur ou d'une case mémoire

DEC : décrémentation d'un accumulateur ou d'une case mémoire

INC : incrémentation d'un accumulateur ou d'une case mémoire.

NOP : pas d'opération, incrémentation du compteur-programme.

COM : complémentation d'un accumulateur ou d'une case mémoire.

NEG : complément à 2 d'un accumulateur ou d'une case mémoire.

Les instructions de transfert des données

LD : chargement des registres internes du CPU à partir d'une case mémoire.
ST : chargement d'une case mémoire à partir des registres internes du CPU.
PSH : empilement de (s) registre(s) sur la pile.
PUL : dépilement de (s) registre(s) depuis la pile.
EXB : échange du contenu de deux registres.
TFR : transfert de registre à registre.

Intructions de test et de branchement

(L) BCC ou (L) BHS : branchement si pas de retenue
(L) BCS ou (L) BLO : branchement si retenue
(L) BEQ : branchement si égale à zéro
(L) BNE : branchement si différent de zéro
(L) BGE : branchement si supérieur ou égal (signé)
(L) BLT : branchement si inférieur (signé)
(L) BGT : branchement si supérieur (signé)
(L) BLE : branchement si inférieur ou égal (signé)
(L) BHI : branchement si supérieur (non signé)
(L) BLS : branchement si inférieur ou égal (non signé)
(L) BHI : branchement si négatif
(L) BPL : branchement si positif
(L) BVC : branchement si pas de débordement
(L) BVS : branchement si débordement

Instruction de saut et de branchement

(L) BRA : branchement inconditionnel
(L) BRN : non branchement
(L) BSR : branchement à sous-programme
JMP : saut inconditionnel à une adresse effective
JSR : saut à un sous-programme
RST : retour de sous-programme

Instruction opérant sur les pointeurs

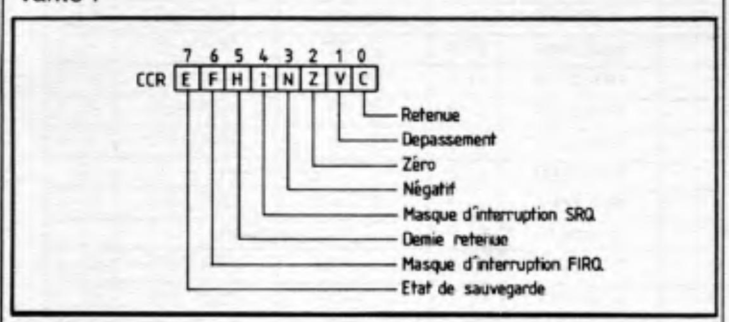
LEA : chargement d'un registre avec une adresse effective

Traitement des interruptions

CWAI : validation puis attente d'une interruption
SYNC : synchronisation du logiciel avec une ligne d'interruption
RTI : retour de sous-programme d'interruption
SWI1/SWI2/SWI3 : interruptions logicielles.

LES FLAGS

Les flags (drapeau en français) sont les bits du registre d'état (CCR) du Microprocesseur.
La fonction de chacun des bits de ce registre est la suivante :



Indicateurs arithmétiques

Ce sont les bits 0, 1, 2, 3 et 5 du registre d'état (CCR). Ils sont positionnés en fonction du résultat des instructions qui manipulent des données.
- Rôle du bit C (Carry = retenue)
Ce bit est positionné à 1 ou à 0 lors d'une opération arithmétique.
- Rôle du bit V (débordement en Complément à deux)
Ce bit est positionné si le résultat d'une opération arithmétique en complément à deux déborde.
- Rôle du bit Z (zéro)
Ce bit est positionné lorsque le résultat de l'opération précédente est nul.
Attention, cet indicateur est également positionné par les opérations de chargement (ex. LDA) et de stockage (ex. STA).
- Rôle du bit N (négatif)
Ce bit est positionné à la valeur du bit de poids fort du résultat d'une opération. En effet, un nombre en complément à deux est négatif si le bit de poids le plus fort est à 1.
- Rôle du bit H (Half carry → demi-retenue)
Ce bit est positionné à 1 si lors d'une opération une retenue passe du bit 3 au bit 4.
Ce bit est l'indicateur de retenue du bit 3.

Les indicateurs d'interruption

Les indicateurs 4, 6 et 7 sont liés au fonctionnement en interruption.
- Rôle du bit I (Interrupt Mask → masque d'interruption)
Ce bit est positionné par l'utilisateur à 1 et interdit le traitement des interruptions IRQ.
- Rôle du bit F (Fast Interrupt Mask → masque d'interruption rapide)
Ce bit a le même rôle que le précédent mais pour les interruptions FIRQ.
- Rôle du bit E (sauvegarde des registres dans la pile)
Ce bit est l'indicateur de sauvegarde des registres dans la pile.
E = 1 → tout le contexte est sauvegardé
E = 0 → une partie seulement des registres est sauvegardée.

LES MODES D'ADRESSAGE

Mode d'adressage inhérent

Ce mode d'adressage est appelé «inhérent» du fait que le microprocesseur sait automatiquement quelle est l'opération à effectuer ainsi que les registres qui sont concernés.

Exemple :

Instruction	Opération effectuée	Registres
ABX code 3A	addition	B + X → X
ASLA code 48	Décalage à gauche	A
ASLB code 58		B
ASRA code 47	Décalage à droite	A
ASRB code 57	•	B
CLRA code 4F	Remise à zéro	A
CLRB code 5F		B
COMA code 43	Complémentation	A
COMB code 53		B
Etc...		

Pour illustrer ce mode d'adressage, je vous propose de réaliser ce qui suit :
- RST → «–»
- REG b → entrer 01 modifier le contenu du registre b
- Reg X → entrer 0009 modifier le contenu du registre X.
Puis entrer le programme suivant :

```
0100    ABX    3A    Addition de B + X

0101    SWI    3F    Retour au moniteur.
```

Lancez maintenant le programme par RST, GO 0100. Vous pouvez maintenant vérifier le contenu du registre X. Je vous parie qu'il y a 0009.
L'instruction ABX est effectivement l'addition du registre B avec X et le résultat est mis dans X.

Comment travaille l'unité centrale avec les mémoires ?

Pour que l'unité centrale «s'y retrouve», il faut lui indiquer dans les instructions codées :
– soit où aller chercher les données à traiter ;
– soit où aller stocker les résultats du traitement.
Dans ce but, le circuit 6809 possède dix modes d'adressage, c'est-à-dire dix façons de coder les adresses, ce qui fait de lui le plus puissant des microproceseurs 8 bits. Il n'est pas question d'étudier dans le détail chacun des dix modes d'adressage. Nous ne présenterons que certains d'entre eux. Les ouvrages cités en bibliographie à la fin permettront de compléter cette introduction.

Programme n°2 : Adressage étendu

La façon la plus simple de donner une adresse est de l'écrire «en clair», c'est-à-dire à l'aide d'un nombre de quatre chiffres hexadécimaux : c'est le mode d'adressage «étendu».
Exemple : transférer une donnée de la case-mémoire n°0210 à la case-mémoire n°0230. En se rappelant que ce transfert ne peut avoir lieu directement mais que la donnée

doit transiter par l'unité centrale, on peut illustrer ce processus de la façon suivante (fig. 15).

L'ordinogramme et le programme correspondants sont simples à écrire :

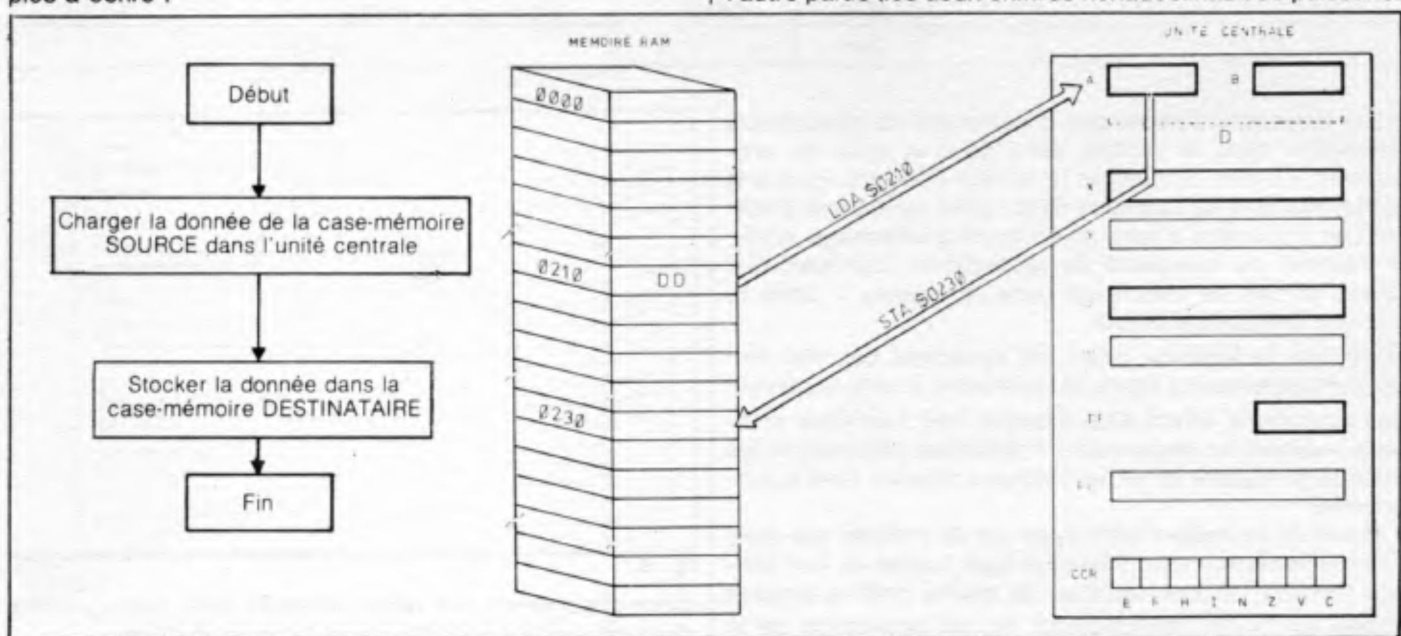


Fig. 15 : Transition des données.

Adresse-programme	Langage machine code-instruction	Langage Assembleur symbolique	opération	opérandes	Commentaires
Ø Ø 2 Ø	B 6 Ø 2 1 Ø		LDA	$Ø210	Amène la donnée dans l'unité centrale
Ø Ø 2 3	B 7 Ø 2 3 Ø		STA	$Ø230	Stockage dans nouvel emplacement-mémoire
Ø Ø 2 6	3 F		SWI		Arrêt - Retour au moniteur (visu registres)
					Fin

Il ne vous reste plus qu'à entrer le programme en code-machine, à le lancer et à vérifier le transfert de la donnée en comparant le contenu des cases-mémoires 0210 et 0230 avant et après exécution du programme.

Remarques

1. Pour trouver le code correspondant aux instructions de chargement et stockage en mode d'adressage étendu, il suffit de consulter le tableau 1.
2. Avec le Microkit 09, seul le langage machine (code héxadécimal) est utilisable.
3. En langage assembleur, une valeur écrite en hexadécimal est précédée du symbole $ (dollar).

Programme 3, 4 et 5 : Adressage direct

Un autre mode d'adressage consiste à n'écrire dans le programme qu'une partie du numéro de l'adresse (les deux chiffres héxadécimaux de puissance inférieure, à droite), l'autre partie (les deux chiffres héxadécimaux de puissance supérieure, à gauche) étant contenus dans un registre dit de «page directe» (DPR «Direct Page Registre»). Il suffira alors à la machine de rassembler les deux parties pour reconstituer l'adresse complète. Dans ce mode d'adressage, les 64K, soit 65 536 numéros de cases-mémoires sont ainsi repérés par 256 pages de 256 numéros chacune.

Programme 3 :

Additionner deux nombres héxadécimaux entrés au clavier, à l'aide du programme-moniteur dans les cases-mémoires n° 0240 et 0241. Stocker le résultat en 0242.

Pour cela, il nous faut charger les deux chiffres (le nombre 02 de la partie supérieure de l'adresse) 0 puis 2 dans le «registre de la page» en utilisant le petit programme suivant. Consulter le tableau 1 pour retrouver les codes utilisés.

Adresse programme	Langage machine code-instruction	Langage Assembleur symbolique	opération	opérandes	Commentaires
Ø Ø 3 Ø	C 6 Ø 2		LDB	#$Ø2	Charger "immédiatement" la valeur 02 dans le registre B
Ø Ø 3 2	1 F 9 B		TFR	B,DP	La transférer pour initialiser le registre de page

Ø Ø 3 4	9 6 4 Ø		LDA	<$4Ø	Chargement du 1er nombre dans unité centrale
3 6	9 B 4 1		ADDA	<$41	Addition avec le 2e nombre
3 8	9 7 4 2		STA	<$42	Stockage du résultat
3 A	3 F		SWI		

Il faut noter qu'il n'existe pas d'instruction de chargement «immédiat» dans le registre DP ; dans la suite du programme, il suffira de charger le nombre correspondant à la partie inférieure de l'adresse dans l'unité centrale et d'utiliser une instruction d'addition en mode d'adressage direct, la machine se chargeant de reconstituer l'adresse. Les codes utilisés se retrouvent dans le tableau 1 dans la colonne «adressage direct».

Noter que l'adressage direct est symbolisé par <en langage Assembleur. La figure 16 représente le séquencement des opérations effectuées. Chaque fois, l'adresse complète «source» ou «destination» s'obtient en rassemblant les chiffres du registre DP et les chiffres contenus dans le programme.

L'intérêt de ce mode d'adressage est de n'utiliser que deux chiffres hexadécimaux d'adresse (soit 1 octet de huit chiffres binaires, chaque «quartet» de quatre chiffres binaires codant un chiffre hexadécimal), ce qui économise de la place pour des opérations concernant une zone-mémoire dont les numéros d'adresse sont situés dans la même «page».

Programme 4

Additionner deux nombres décimaux entrés par clavier, à l'aide du programme-moniteur. Ce programme implique :

- de rentrer des nombres comportant seulement des chiffres de 0 à 9, c'est-à-dire des nombres décimaux ;
- d'effectuer dans le programme après l'addition, un ajustement décimal, à l'aide de l'instruction DAA, de façon à obtenir le résultat en décimal. A noter que DAA ne fonctionne que derrière une instruction d'addition et n'agit que sur l'accumulateur A.

Nous vous proposons d'essayer d'écrire ce programme seul, avant de le comparer au listing des programmes présentés.

Programme 5

Additionner deux nombres de 16 bits (2 octets). Il faut alors utiliser deux cases-mémoires pour écrire chaque nombre, et l'accumulateur D (composé par l'assemblage des accumulateurs A et B) pour effectuer l'opération.

Programmes 6, 7 : Adressage immédiat

L'adressage «immédiat» (que nous avons utilisé sans l'expliciter à la première ligne du programme 3) consiste à introduire directement une valeur, indiquée dans le programme, dans un registre spécifié par le code-instruction.

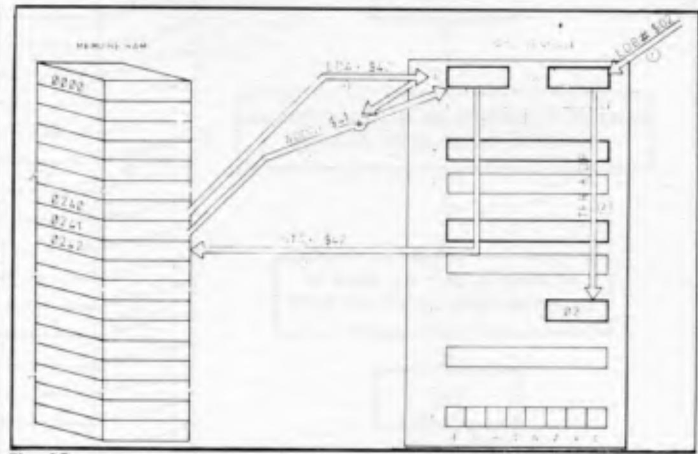


Fig. 16.

Programme 6

Additionner deux nombres hexadécimaux entrés par programme dans les cases-mémoires 0240 et 0241.

Nous pouvons stocker les nombres en utilisant l'adressage immédiat (symbolisé par # en langage Assembleur) et l'adressage direct (symbolisé par <en langage Assembleur) suivant le programme ci-dessous et la figure 17.

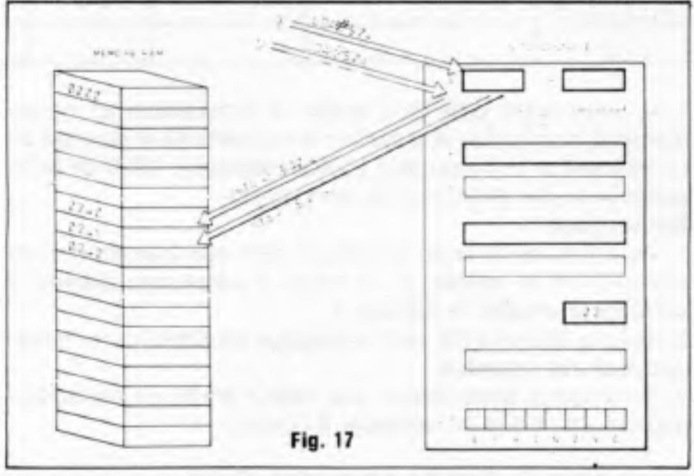


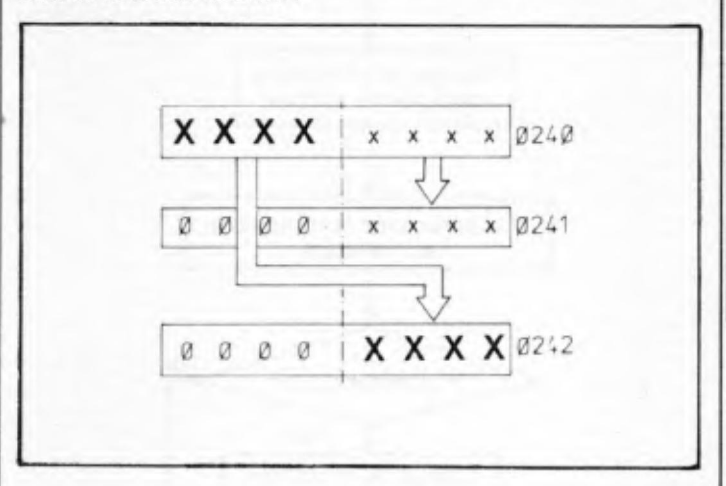
Fig. 17

Adresse-programme	Langage machine code-instruction	Langage Assembleur symbolique	opération	opérandes	Commentaires
Ø Ø 6 Ø	8 6 Ø 3		LDA	#$03	Stockage du 2e nombre
6 2	9 7 4 Ø		STA	<$40	
6 4	8 6 Ø 4		LDA	#$04	Stockage du 1er nombre
6 6	9 7 4 1		STA	<$41	

La suite du programme d'addition est similaire à la deuxième partie du programme 3.

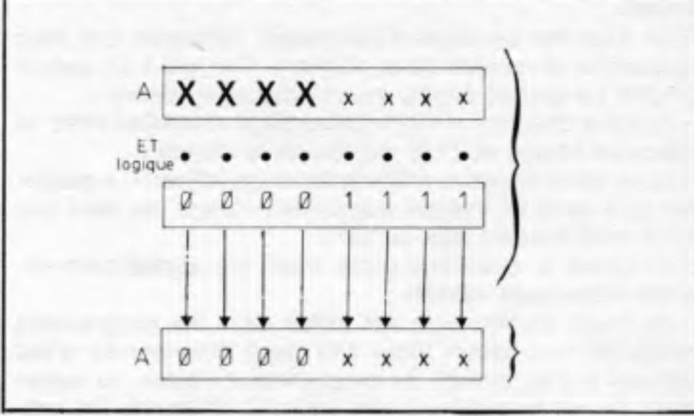
Ø Ø 6 8	9 6 4 Ø		LDA	<$4Ø	
6 A	9 B 4 1		ADDA	<$41	
6 C	9 7 4 2		STA	<$42	
6 E	3 F		SWI		

Programme 7
Séparer l'octet (huit chiffres binaires) inscrit en deux quartets (quatre chiffres binaires) à ranger en 0241 et 0242 suivant le schéma suivant :



Pour cela, après avoir chargé le nombre dans le registre A (ou B), nous allons utiliser des instructions d'opération logique avec adressage immédiat, et de décalage de chiffres binaires. Mais écrivons d'abord l'ordinogramme puis le programme :

L'opération de masquage se réalise de la façon suivante :

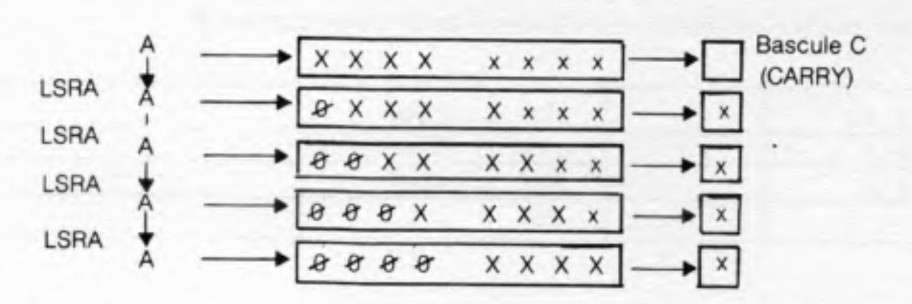


On effectue un ET logique entre chaque chiffre binaire de l'octet mis dans A et le chiffre binaire correspondant de la valeur introduite en adressage immédiat avec l'instruction. ANDA # % 00001111. Le symbole % indique que la valeur est écrite en binaire dans le programme en langage Assembleur. On pourrait l'écrire en hexadécimal ($OF).
On a donc «masqué» le quartet de droite en le remplaçant par des 0.

L'instruction du décalage LSRA permet de décaler le quartet de gauche sur la droite et de le remplacer par 0.

Début
Charger le nombre dans l'unité centrale
Masquer le quartet de gauche
Stocker le quartet de droite restant
Recharger le nombre
Décaler le quartet de gauche sur la droite et le remplacer par des Ø
Stocker ce quartet
Fin

Adresse-Programme	Langage machine code-instruction	Langage Assembleur symbolique	opération	opérandes
Ø Ø 7 Ø	9 6 4 Ø		LDA	<$4Ø
7 2	8 4 Ø F		ANDA	#%00001111
7 4	9 7 4 1		STA	<$41
7 6	9 6 4 Ø		LDA	<$4Ø
7 8	4 4		LSRA	
7 9	4 4		LSRA	
7 A	4 4		LSRA	
7 B	4 4		LSRA	
7 C	9 7 4 2		STA	<$42
7 E	3 F		SWI	



Programme 8 et 9 : Adressage relatif ou la «Puce sauteuse»

Pour expliciter ce mode d'adressage, supposez que vous demandiez le numéro de la chambre d'un ami à un garçon d'hôtel. Le garçon pourra vous l'indiquer en disant :

– «c'est la chambre n°1221» (adressage **«étendu»**) avec 12 indiquant l'étage et 21 le numéro de la chambre ;

– ou «c'est la chambre n°21» (adressage **«direct»**) à supposer qu'il vous ait indiqué auparavant l'étage, ou bien que vous vous trouviez déjà au 12e ;

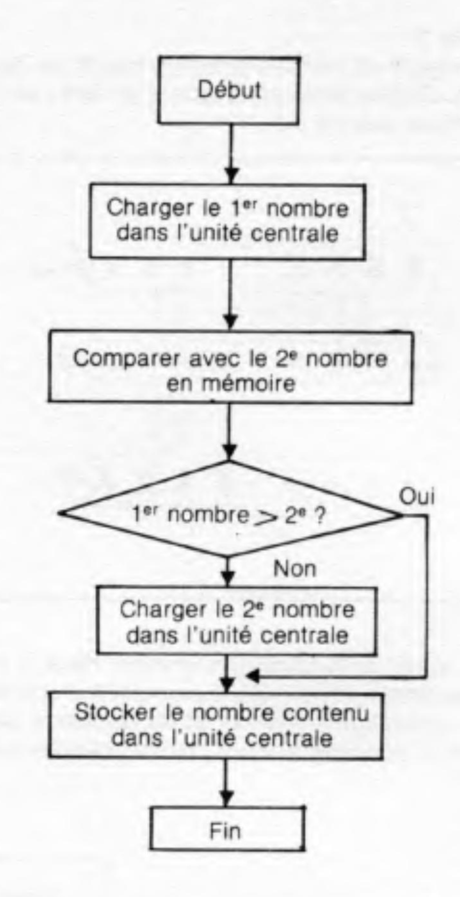
– ou «c'est la quatrième porte avant (ou après) celle-ci», c'est l'adressage «relatif» ;

– ce mode d'adressage est utilisé dans les programmes lorsqu'on veut sauter (pour une puce programmée **c'est naturel !**) d'un endroit du programme à l'autre, ou autrement dit «se brancher» à une adresse différente. On indiquera alors non pas en absolu l'adresse de destination mais la distance qui vous en sépare. Les deux exemples qui suivents illustrent ce mode.

Programme 8

Trouver le plus grand des deux nombres stockés en 0240 et 0241 et le ranger en 0242.

On va évidemment utiliser ici une instruction de comparaison, mais suivant le résultat de la comparaison, il faudra ranger soit l'un, soit l'autre des deux nombres. C'est ce qu'illustre l'ordinogramme ci-dessous :



On voit donc que si l'on répond positivement à la question «le 1er nombre est-il plus grand que le 2e ?», il faut alors se brancher plus loin dans le programme et sauter la partie du programme exécutée dans le cas où l'on répondrait négati-

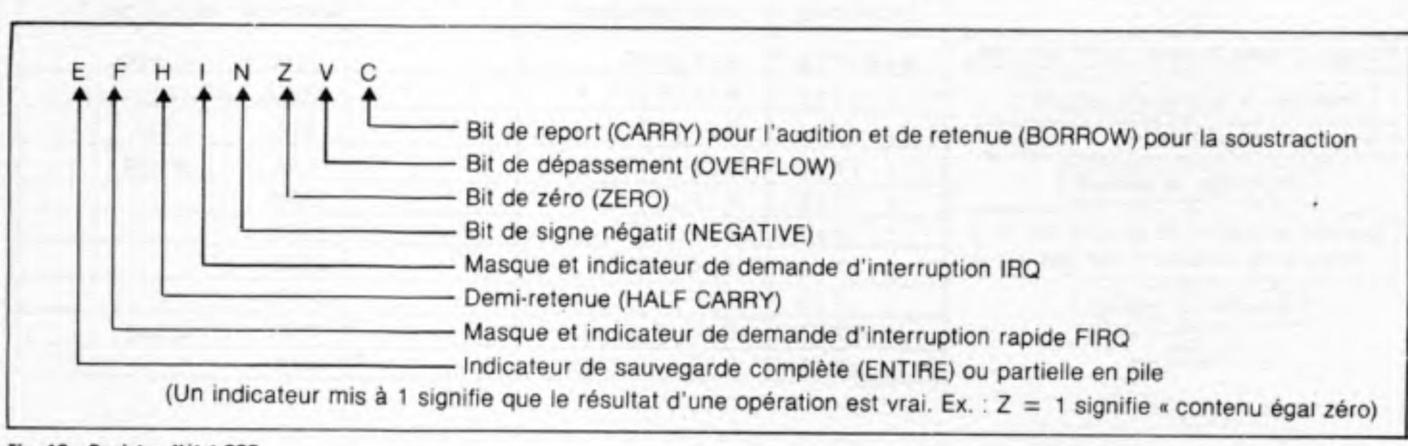


Fig. 18 : Registre d'état CCR.

vement. Mais qui nous indique le résultat positif et négatif de la comparaison ? De combien de pas devons-nous sauter si c'est positif ?

Il existe dans l'unité centrale des bascules-indicateurs, nous donnant des renseignements sur le résultat d'opérations. Ces bascules sont rassemblées dans un registre CCR («Code Condition Register»), appelé aussi «registre d'état et des indicateurs d'états» et décrit en figure 18.

Pratiquement, lorsqu'on écrit une instruction demandant un branchement dans le programme en fonction du résultat d'une opération, la machine va tester les indicateurs C, V, Z et N.

Les indicateurs I, F et E sont utilisés dans les programmes d'interruption.

Ecrivons maintenant le programme de comparaison :

Adresse-programme	Langage machine code-instruction		Langage Assembleur symbolique		Commentaires
Ø Ø 8 Ø	9 6 4 Ø		LDA	<$4Ø	Charger le 1er nombre dans l'unité centrale
Ø Ø 8 2	9 1 4 1		CMPA	<$41	Le comparer au 2e
Ø Ø 8 4	2 4 □ □		BHS	□□	Si 1er nombre plus grand se brancher
Ø Ø 8 6	9 6 4 1		LDA	<$41	Charger le 2e nombre
Ø Ø 8 8	9 7 4 2	□ □	STA	<$42	Stocker le nombre le plus grand
8 A	3 F		SWI		Fin

On a volontairement laissé en blanc la valeur du saut à effectuer dans le programme, valeur à écrire après l'instruction BHS qui ordonne le branchement dans le cas où le résultat de la comparaison (qui équivaut à une soustraction) est positif ou nul.

Comment calculer le déplacement en adressage relatif ?

1. Lorsque la machine a décodé l'instruction BHS suivie de la valeur du déplacement que nous allons calculer, le compteur-programme qui indique où on est dans le programme pointe alors l'adresse de l'instruction suivante : soit 0086. Il faut donc faire sauter le compteur programme de deux pas vers l'avant. La valeur du déplacement à inscrire ici à l'adresse 0085 est donc de 0 2.

2. On peut imaginer un compteur situé dans les emplacements laissés en clair □ □ à l'adresse 0085, et initialisé à FF. Pour atteindre l'adresse de destination, ici 0088, on va s'y transférer en incrémentant à chaque passage d'adresse le compteur de 1.

Ce qui donne l'évolution suivante : 0085 → 0086 → 0087 → 0088. Compteur : FF → 00 → 01 → 02.

3. Solution «relaxe» : il existe dans le programme-moniteur de la maquette Micro Kit 09 un «calcul automatique du déplacement» pour ce mode d'adressage.

Ce calcul s'effectue à l'aide de la touche ofs «offset».

Programme 9

Décompter de 255 à 000.

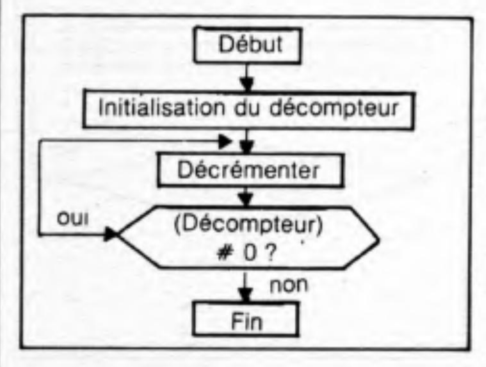
L'ordinogramme et le programme sont très simples et reproduits ci-dessous. Il suffit de créer une «boucle» logicielle tant que le contenu du registre-décompteur n'est pas nul.

– Premier calcul du déplacement :

Après décodage de l'instruction demandant un branchement à l'instruction décomptage tant que le décompteur n'est pas nul, le compteur d'adresses-programme (PC) se trouve en 0095, il devra sauter en 0092 soit un saut en arrière de – 3. Mais comment écrire – 3 en héxadécimal ? En utilisant le code dit «complément à 2». Le signe « – » sera codé par le chiffre binaire 1, placé le plus à gauche puis au lieu d'écrire la valeur 3 on écrira son complément à 2^7, soit 128 – 3 = ce qui s'écrit en binaire :

1 111 1101
signe — complément à 2^7 de 3

soit FD en hexadécimal.



Adresse	Langage machine code-instruction		Langage opération	Assembleur symbolique opérandes
Ø Ø 9 Ø	8 6 F F		LDA	$FF
Ø Ø 9 2	4 A		DECA	
Ø Ø 9 3	2 6 □ □		BNE	□ □
Ø Ø 9 5	3 F		SWI	

– Deuxième calcul de déplacement :
En reprenant la méthode du compteur placé en 0094, et initialisé à FF puis déplacé et décrémenté progressivement jusqu'en 0092 on retrouve facilement la valeur F D.
– Troisième calcul :
avec la touche offs.

Programmes 10 à 12 : Adressage indexe

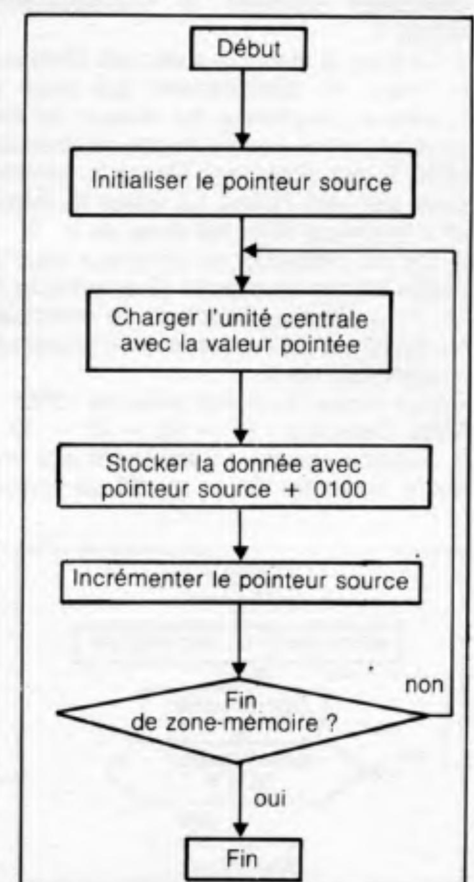
Un autre mode d'adressage, très varié, consiste à calculer l'adresse effective de la case-mémoire, où on va soit chercher soit stocker une donnée, à partir d'une adresse de base à laquelle s'ajoute un déplacement.
Ainsi, si l'adresse de base est 0200, pour aller en 0240, il suffira d'indiquer un déplacement de $40. L'adresse de base sera inscrite dans un registre X, Y ou même U ou S ou éventuellement D.

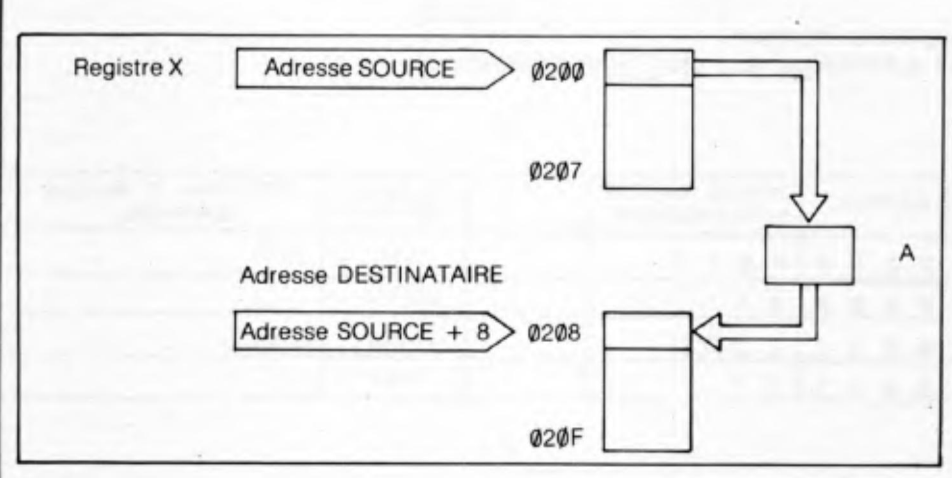
Programme 10 : Recopie de zone-mémoire

L'on désire recopier les données inscrites entre les adresses 0200 à 0207 dans une zone-mémoire située entre 0208 et 020F. Le principe de cette recopie est schématisé ci-dessous :
On note que :
. la donnée transite par le registre A ;
. l'adresse de base (qui est l'adresse de début de la zone-mémoire source) est inscrite dans le registre X appelé aussi registre-index, car «**pointant**» une adresse ;
. l'adresse effective du destinataire, qui correspond au début à la première adresse de la zone-mémoire destinataire est donc égale à la source +8.
Nous donnons ci-contre l'ordinogramme puis le programme en langage assembleur symbolique. Amusez-vous à l'écrire en langage machine en vous aidant des remarques qui suivent, avant de le comparer à celui proposé.

Remarques (1)
Ce mode d'adressage est **indexé à déplacement nul.** L'adresse effective est égale à : (l'adresse base/inscrite dans X) + (0). Cette instruction se code en s'aidant du tableau 1 et du tableau 2 (reproduit ci-dessous).
LDA : en adressage indexé se code A6 ;
X : signifie que l'adresse de base est inscrite dans X et qu'on lui ajoute un déplacement nul. La première ligne du tableau 2 (mode non indirect) nous indique le code 1RR00100 où RR est le code du registre choisi, soit 00 pour X. Le code 84 en hexadécimal ainsi obtenu est appelé le «post-octet» (car venant après le ou les octet(s) du code opération de l'instruction), et spécifie le mode d'adressage indexé.
(2) : L'adresse effective de stockage est ici obtenue par un adressage **indexé à déplacement constant** égal à 8 (écrit ici en décimal, non précédé du symbole $).
STA : se code en adressage indexé A7 ;

Adresse-programme	Langage machine code-instruction		Langage assembleur symbolique Opération	Opérandes	Remarques
0100			LDX	#$0200	
			LDA	,X	(1)
			STA	8,X	(2)
			LEAX	1,X	(3)
			CMPX	#$0208	
			BNE	□ □	(4)





Type	Formes	Mode non indirect				Mode indirect			
		Syntaxe assembleur	Post-octet code OP	~	N	Syntaxe assembleur	Post-octet code OP	~	N
Déplacement constant à à partir de R (signé)	pas de déplacement	, R	1RR00100	0	0	[, R]	1RR10100	3	0
	déplacement 5 bits	d, R	0RRnnnnn	1	0	par défaut — 8 bits			
	déplacement 8 bits	d, R	1RR01000	1	1	[d, R]	1RR11000	4	1
	déplacement 16 bits	d, R	1RR01001	4	2	[d, R]	1RR11001	7	2
Accumulateur utilisé comme déplacement pour le Registre R (déplac. signé)	registre de déplac. A	A, R	1RR00110	1	0	[A, R]	1RR10110	4	0
	registre de déplac. B	B, R	1RR00101	1	0	[B, R]	1RR10101	4	0
	registre de déplac. D	D, R	1RR01011	4	0	[D, R]	1RR11011	7	0
Auto incrémentation/décrémentation du registre R	incrémenté par 1	, R +	1RR00000	2	0	Impossible			
	incrémenté par 2	, R + +	1RR00001	3	0	[,R + +]	1RR10001	6	0
	décrémenté par 1	, -R	1RR00010	2	0	Impossible			
	décrémenté par 2	,--R	1RR00011	3	0	[,--R]	1RR10011	6	0
Déplacement constant à partir de PC	déplacement 8 bits	d, PCR	1XX01100	1	1	[d, PCR]	1XX11100	4	1
	déplacement 16 bits	d, PCR	1XX01101	5	2	[d, PCR]	1XX11101	8	2
Indirect étendu	adresses 16 bits	—	—	—	—	[d]	10011111	5	2

Légende :
R = registres X, Y, U ou S
X = valeur indifférente
~ = nombre de cycles-horloge additionnels
N = nombres d'octets additionnels
d = valeur du déplacement (décimal)
RR = code registre 00 = X
01 = Y
10 = U
11 = S
nnnnn = valeur du déplacement en complément à 2

(d'après documents Motorola/EFCIS).

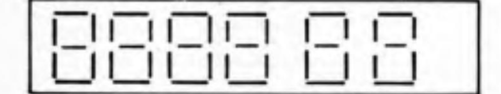
Tableau 2 : Types d'adressage indexé et codage du post-octet.

8,X : se code (voir deuxième ligne du tableau 2).

ORR n nnnn
Code registre — Valeur du déplacement codé en complément à 2

On rappelle que dans le code complément à 2, le chiffre binaire le plus à gauche indique le signe (0 pour +, 1 pour −), et qu'on écrit une valeur positive en code binaire pur et une valeur négative avec son complément à 2^n.
Ici, le déplacement de valeur 8 étant compris entre − 16 et + 15, quatre bits suffisent pour coder sa valeur, plus un bit de signe.
Si la valeur du déplacement est comprise entre − 128 et + 127 il faudra ajouter à l'octet (ou aux deux octets) codant l'opération, et au post-octet (voir troisième ligne du tableau) un octet supplémentaire codant la valeur du déplacement. Voir la colonne N du tableau 2.
Deux octets supplémentaires seront nécessaires pour une valeur de déplacement comprise entre − 32 768 et + 32 767.
Sur la maquette Microkit 09, le calcul du code hexadécimal des post-octets contenant la valeur du déplacement est automatisé.
Exemple : nous voulons coder l'instruction LDA − 17,X (− 17 étant exprimé en décimal).
En cours d'écriture du programme du clavier :
1. Entrer le code A 6 pour «LDA»
2. Lecture du code du post-octet indiquant que l'on utilise le registre d'index X pour un déplacement codé sur 8 bits : en effet, pour coder − 17 en code complément à 2, il faut au moins cinq chiffres binaires significatifs et un bit de signe. On choisit donc la troisième ligne du tableau, correspondant à un déplacement codé avec 8 bits :

1RR0 1000 soit avec le registre d'index X, 1000 1000 = $88.
− Entrer 8 8.
3. Calcul automatique du déplacement :
- appuyer sur X puis sur Ofs l'affichage présente alors :



− Préciser le sens du déplacement en appuyant :
soit sur INC pour un déplacement positif → affichage de **P** soit sur Dec pour un déplacement négatif → affichage de **n** ... dans cet exemple, on appuie donc sur Dec.
− Entrer la valeur du déplacement 00017.
− Appuyer sur GO.

Le programme-moniteur calcule alors le code du déplacement, soit ici EF, et le place dans la (les) case(s) mémoire suivante(s) de notre programme. L'instruction LDA-17, X est donc codée avec : A6 88EF. Pas de panique, si vous vous trompez dans la procédure de codage, la maquette devrait afficher :

(comme Erreur)

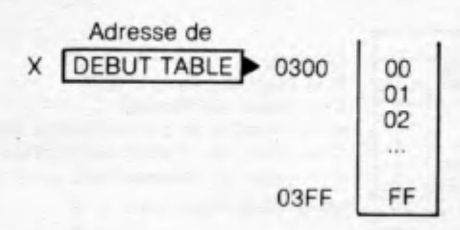
3. Pour incrémenter le contenu du registre pointeur X, c'est-à-dire pour incrémenter l'adresse de base, on utilise l'instruction de calcul d'une adresse effective.
LEA, associée à un adressage indexé à déplacement constant qui s'é crit en langage symbolique LEAX 1,X et qui signifie :
LEAX : «Charge dans X l'adresse effective...»
1,X : «... obtenue en prenant le contenu de X et en lui ajoutant 1».
Le code de LEAX est 30. Le code de 1,X se lit à la deuxième ligne du tableau (mode non direct).

4. La valeur du déplacement pour le branchement s'obtient suivant les indications données pour les programmes 8 et 9.

La figure ci-contre résume les types utilisés d'adressage indexé.

Question «blanche» : Faire un programme qui recopie de 01F8 à 01FF les contenus de la zone 0200 à 0207.

Programme 11
Inscrire des objets par ordre croissant dans une table-mémoire de 0300 à 03FF.
Il s'agit d'inscrire par programme :
$ 00 dans la case-mémoire 0300



$ 01 dans la case-mémoire 0301

....

$ FF dans la case-mémoire 03FF

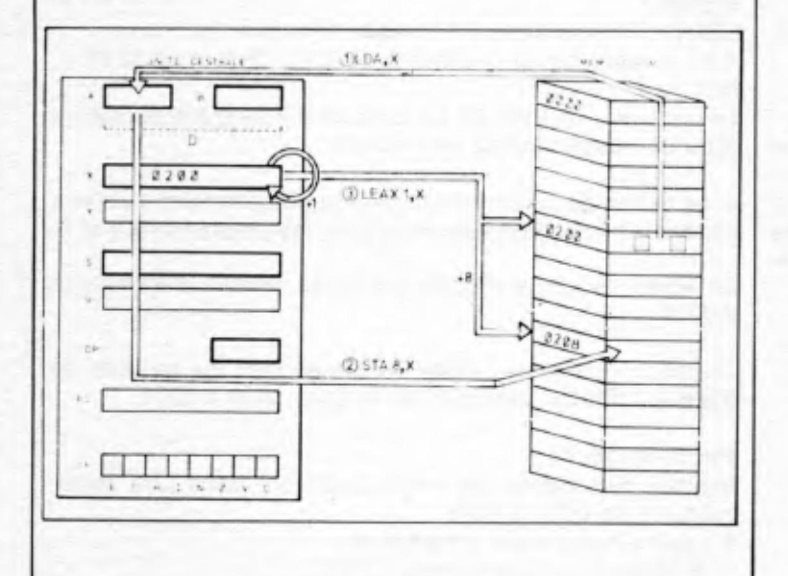
Adresse-programme	Langage machine code - instruction		Langage assembleur symbolique Opération	Opérandes	
0, 1 1, 0			LDX	#$0300	
			CLRA		
			CLRB		
			STB	D,X	(1)
			INCB		
			BNE	□ □	
			SWI		

Remarque (1)

Dans ce mode **d'adressage indexé à déplacement accumulateur** la valeur du déplacement se trouve dans l'accumulateur D. En effet, le déplacement est ici lié à la valeur contenue dans le générateur d'octets.

Question «rouge»
Pourquoi ne pas avoir choisi simplement l'accumulateur A ou B au lieu de D comme valeur du déplacement dans l'instruction STB, D,X ?

Essayez alors votre programme modifié, avec une table-mémoire située entre 0500 et 05FF puis vérifiez le contenu de cette zone et de la zone 0480 à 04FF. Que s'est-il passé ?



Programme 12 :
Faire un décompteur de temps utilisant les registres X et A, initialisés respectivement à FFFF et FF. Calculer la durée totale de décomptage.

. Indications :

1. **L'utilisation de deux décompteurs se fait à l'aide de boucles logicielles encastrées**, selon l'ordinogramme ci-contre
2. Le calcul du temps mis pour le décomptage se fait en utilisant le tableau 1 où est indiqué dans la colonne ~ le nombre de cycles horloge que dure une instruction, sachant que le cycle-horloge du Microkit 09 est de 1 µs.

Ex : Charger le registre X avec la valeur FFFF en adressage immédiat dure 3 µs tandis que l'exécution de SWI dure 19 µs.

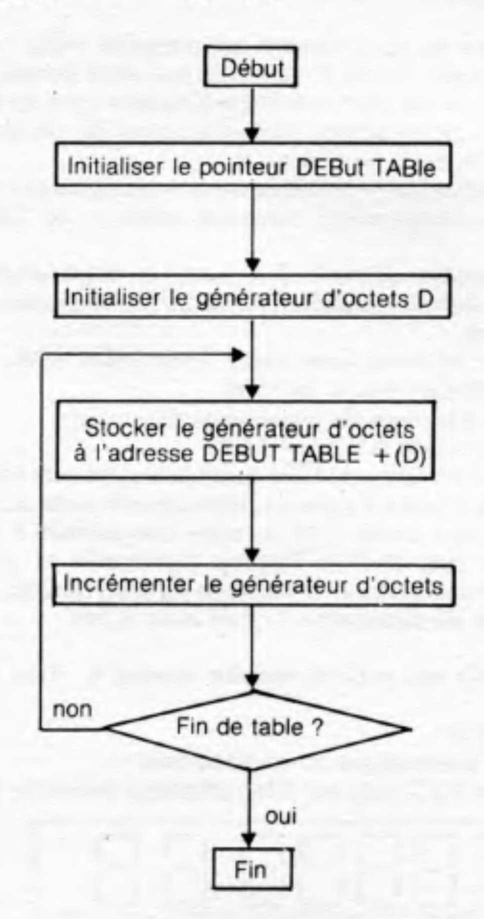
EX. : Si la boucle 2 est parcourue FF fois, le temps d'exécution de cette boucle sera 255 (2 + 3) µs, 2 µs correspondant à l'instruction DEC, 3 µs correspondant à l'instruction de branchement BNE.

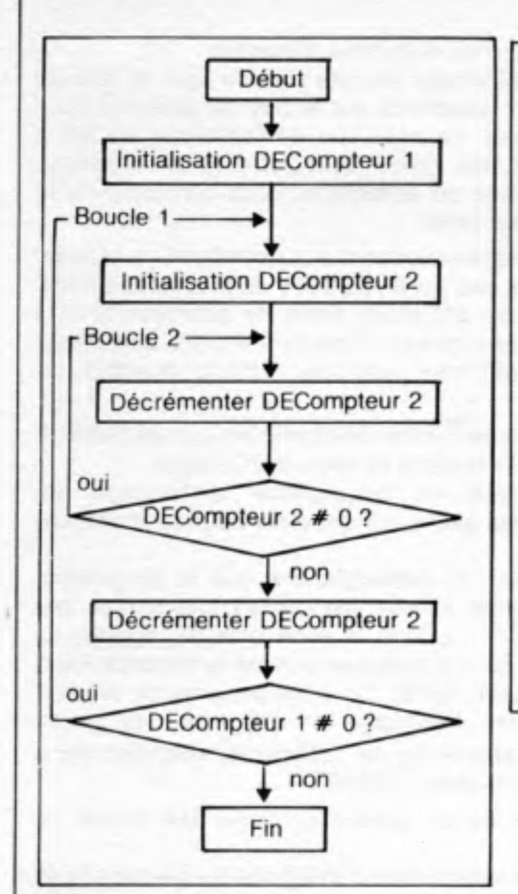
3. Si on utilise comme décompteur le registre X, il n'existe pas d'instruction «DEX» ! Mais, par contre, il existe une instruction LEAX (Load Effective Adress) qui permet de charger une valeur dans X. Cette valeur peut être l'ancienne valeur de X diminuée de 1.

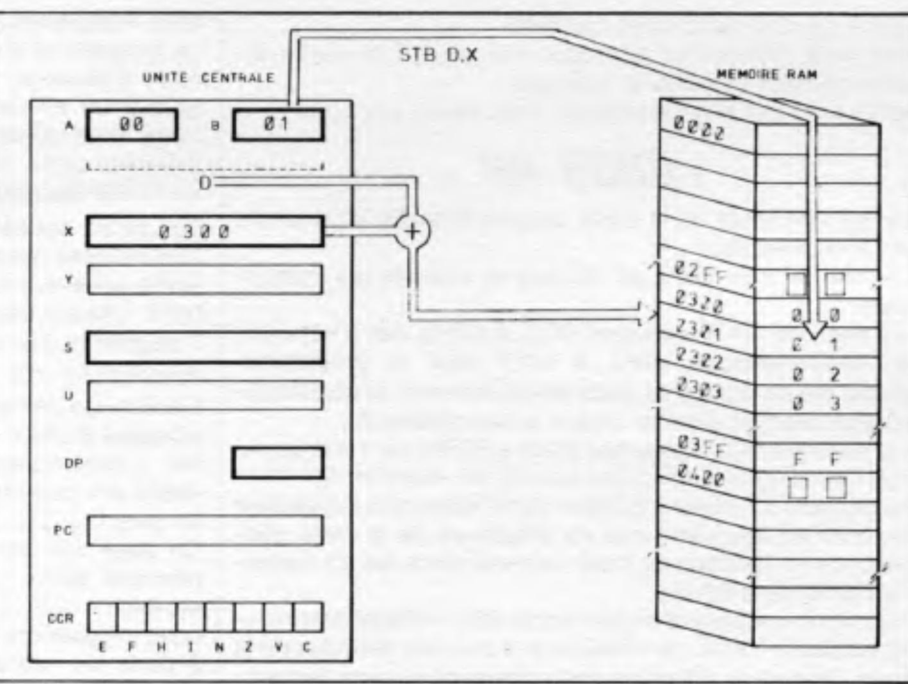
Cette instruction s'écrit en langage symbolique :

LEAX , – X

et se code : 30 , 82







Adresse-programme	Langage machine code - instruction				Langage assembleur symbolique	
	1er octet	2e octet	3e octet		Opération	Opérandes
0 0 0 0	C C	7 D	7 F		LDD	#$7D7F
0 3	F D	0 7	F A		STD	<$07FA
0 6	C C	7 E	3 F		LDD	#$7E3F
0 9	F D	0 7	F C		STD	< $07FC
0 C	C C	E 3	6 B		LDD	#$E36B
0 F	F D	0 7	F E		STD	< $07FE
1 2	B D	E 0	7 B		▶JSR	$E07B
1 5	2 0	F B			BRA	
1 7						

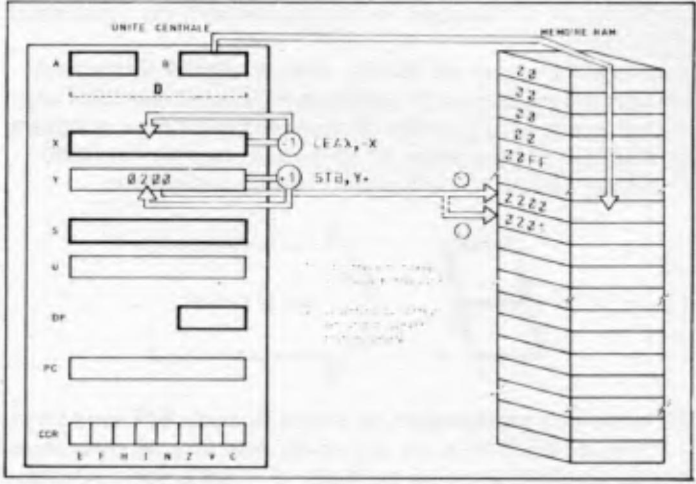


Fig. 19 : Auto-incrémentation.

Son temps d'exécution est de :

$$(4 + 2)\ \mu s.$$

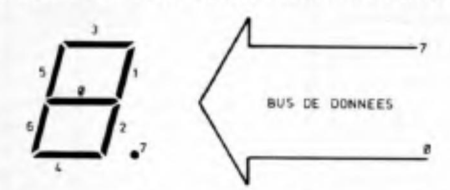
Elle signifie
LEAX : «Charge dans X l'adresse effective...»
, – X : «... obtenue 1°» en prédécrémentant le contenu de X, puis 2° en rechargeant cette nouvelle valeur».
Cette instruction utilise le mode d'**adressage indexé à autoprédécrémentation.** Il existe aussi des possibilités de double prédécrémentation, (-- R), et de post-incrémentation automatiques, R + et ,R + +. Se reporter au tableau 2 pour le codage de ce type d'adressage schématisé en figure 19.

4. Le listing complet du programme et le temps mis pour son exécution se trouvent en fin d'article... Mais essayez d'abord de le mettre au point par vous-même.
Vérifier le temps indiqué avec votre montre en chronométrant le temps écoulé entre le lancement du programme (l'affichage du Microkit 09 s'éteint) et le moment où il se rallume (après avoir rencontré l'interruption SWI).
Comme cela apparaît dans le tableau 2, il existe d'autres types d'adressage indexé, sans parler du mode indirect. Dans ce mode, l'unité centrale «va chercher chez Dupont l'adresse de Durand».
Nous allons maintenant voir comment l'unité centrale travaille non seulement avec les mémoires mais avec les périphériques, via le coupleur d'entrée/sortie (circuit 6821 en

haut à gauche de la carte centrale).
Pour cela, introduisez le programme suivant ci-contre en mémoire RAM et faites-le exécuter.
Après avoir lancé le programme, vous devez voir apparaître

« 6809 H.P »

sur les afficheurs de la carte périphérique. Ce programme fait donc travailler :
– l'unité centrale 6809 qui décode et exécute les instructions ;
– la mémoire RAM (adresses 0000 à 07FF), car c'est dans les cases-mémoire 07FA à 07FF que le programme stocke les six octets de code précédemment et consécutivement chargés dans le double accumulateur D ;
– la mémoire ROM (adresses E000 à E7FF), car l'instruction JSR (Jumping to SubRoutin) appelle un «sous-programme» qui démarre à l'adresse E07B et dont l'exécution fait allumer (ou non) les segments des six afficheurs de la carte périphérique en fonction du code contenu dans les six cases-mémoires de la RAM ;
– le circuit coupleur d'entrée/sortie 6821 («Péripheral interface adapter : PIA), car il interface le bus des données de la carte centrale à l'affichage et au clavier de la carte périphérique ;
– les six afficheurs et le circuit de sélection.
La figure 20 schématise le fonctionnement du processus d'affichage.
Avant de l'analyser en détails, voici quelques remarques :
(**1**) Les huit fils du bus de données sont reliés aux sept segments lumineux et au point décimal des afficheurs, à travers le coupleur entrée/sortie et un circuit «buffer» 74LS240.
L'interconnexion est faite selon l'ordre ci-dessous.



Si, sur le bus de données se trouve le code $7F (soit 0111 1111 en binaire), tous les segments vont être allumés, mais non le point décimal. Le symbole 8 sera donc affiché.
Le code $7E (soit 0111 1110 en binaire) fera apparaître le symbole 0 , le segment central n'étant pas allumé.
Il est alors facile de déduire les codes d'allumage 7 segments des symboles hexadécimaux (tableau 3).
Cette table appelée DIGTBL («Digit table») = table des afficheurs) est en fait mémorisée en mémoire ROM à partir de l'adresse E010 mais n'est pas utilisée dans notre programme.
On en déduire facilement le code 7 segments des symboles μ et P (comme «micro-processeur»).
(**2**) Pour des raisons de séquencement programmé (un seul code d'allumage présent à la fois sur le bus de données et destiné à un seul afficheur) et de persistance rétinienne (il suffit de rafraîchir le contenu d'un afficheur d'au moins 25 fois par seconde), les codes d'allumage 7 segments ne sont pas mémorisés en permanence dans chaque afficheur mais multiplexés... d'où économie d'énergie.
Le programme d'affichage placera donc à tour de rôle un code d'allumage 7 segments sur le bus de données pour allumer un afficheur. La sélection de l'afficheur se fait à l'aide d'un code mis dans le registre B du coupleur d'entrée/sortie, code qui activera la sortie correspondante du circuit décodeur 7442.
(**3**) Les six codes correspondant aux six afficheurs doivent être stockés dans une zone réservée de la mémoire RAM. Cette zone-mémoire est située entre les adresses 07FA à 07FF, chaque case-mémoire contenant le code d'allumage 7 segments d'un afficheur particulier selon la répartition ci-dessous (fig. 20).
L'ensemble de ces six cases-mémoires est appelé DISBUF («Display buffer» = registre tampon d'affichage).
(**4**) L'ordinogramme du programme d'affichage de «6809 μP» que nous avons fait exécuter peut se représenter ainsi (fig. 21).
On peut voir dans cet ordinogramme que le programme principal après avoir stocké les codes d'allumage des symboles «6809μP » dans la zone DISBUF, appelle un sous-programme qui est déjà inscrit dans la mémoire ROM à partir de l'adresse E07B. Ce sous-programme va aller chercher les codes d'allumage les uns après les autres pour activer les segments de l'afficheur, sélectionnés à partir d'un code mis dans DISCNT.
Le listing détaillé de ce sous-programme est donné en encadré.
– L'appel du sous-programme d'affichage se fait avec l'instruction JSR $E07B (notez l'adressage étendu).
– Mais comment l'unité centrale sait-elle où retourner dans notre programme principal lorsque le sous-programme d'affichage est exécuté ?
Cela lui est facile car elle a sauvegardé l'adresse de retour, ici 0015, après avoir décodé l'instruction d'appel du sous-programme JSR. Cette adresse, qui était alors contenue dans le compteur-programme (PC) est sauvegardée dans une zone de la mémoire RAM appelée «Pile» («Stack» en anglais), car c'est là qu'on «empile» les données à sauvegarder. C'est dans cette pile que l'unité centrale viendra chercher l'adresse où retrouver (0015) à la fin du sous-programme pour la remettre dans le compteur-programme PC par l'instruction PULS PC ou RTS. Notez que le sous-programme d'affichage sauvegarde aussi les contenus de X, B et A en les empilant par l'instruction PSHS et les récu-

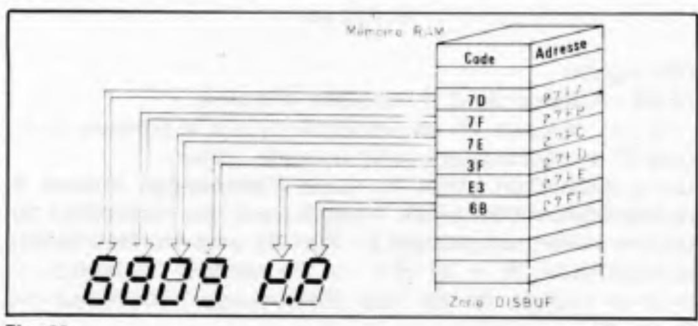


Fig. 20

père en les dépilant par l'instruction PULS, ceci afin d'éviter que leurs valeurs utilisées dans un programme principal soient modifiées.
La figure 22 schématise ces opérations de sauvegarde dans la pile.

Symbole	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Code d'allumage 7 segments	7E	06	5B	1F	27	3D	7D	0E	7F	3F	6F	75	78	57	79	69

Programme principal
(mis en mémoire RAM)

Début
0000
Stocker les 6 codes d'allumage 7 segments dans DISBUF
000F
0012
Appeler le sous-programme d'affichage
0015
Se rebrancher à l'instruction précédente

JSR
PULS,PC
ou RTS
Retour

Sous-programme d'affichage
(en mémoire ROM)

E07B
Initialiser le coupleur entrée/sortie
tous les fils du port A en sortie,
la moitié des fils du port B en sortie
E093
Pointer le 1er code d'allumage 7 segments dans DISBUF
E096
Initialiser le compteur DISCNT de sélection afficheur
Incrémenter DISCNT
Tous afficheurs scrutés ?
oui
non
Sélectionner l'afficheur
Allumer les segments suivant le code dans DISBUF
Temporisation de 1 ms
Se rebrancher pour l'affichage suivant

Fig. 21 : Ordinogramme complet du programme d'affichage.

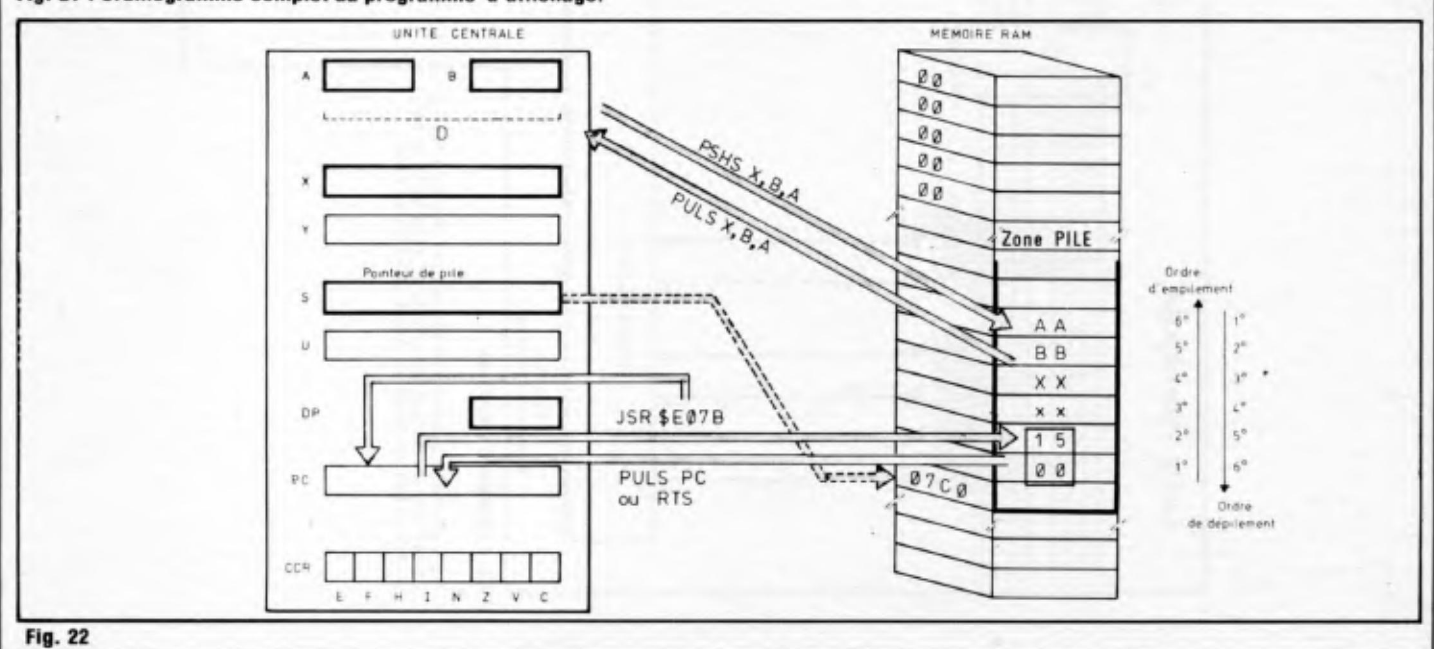


Fig. 22

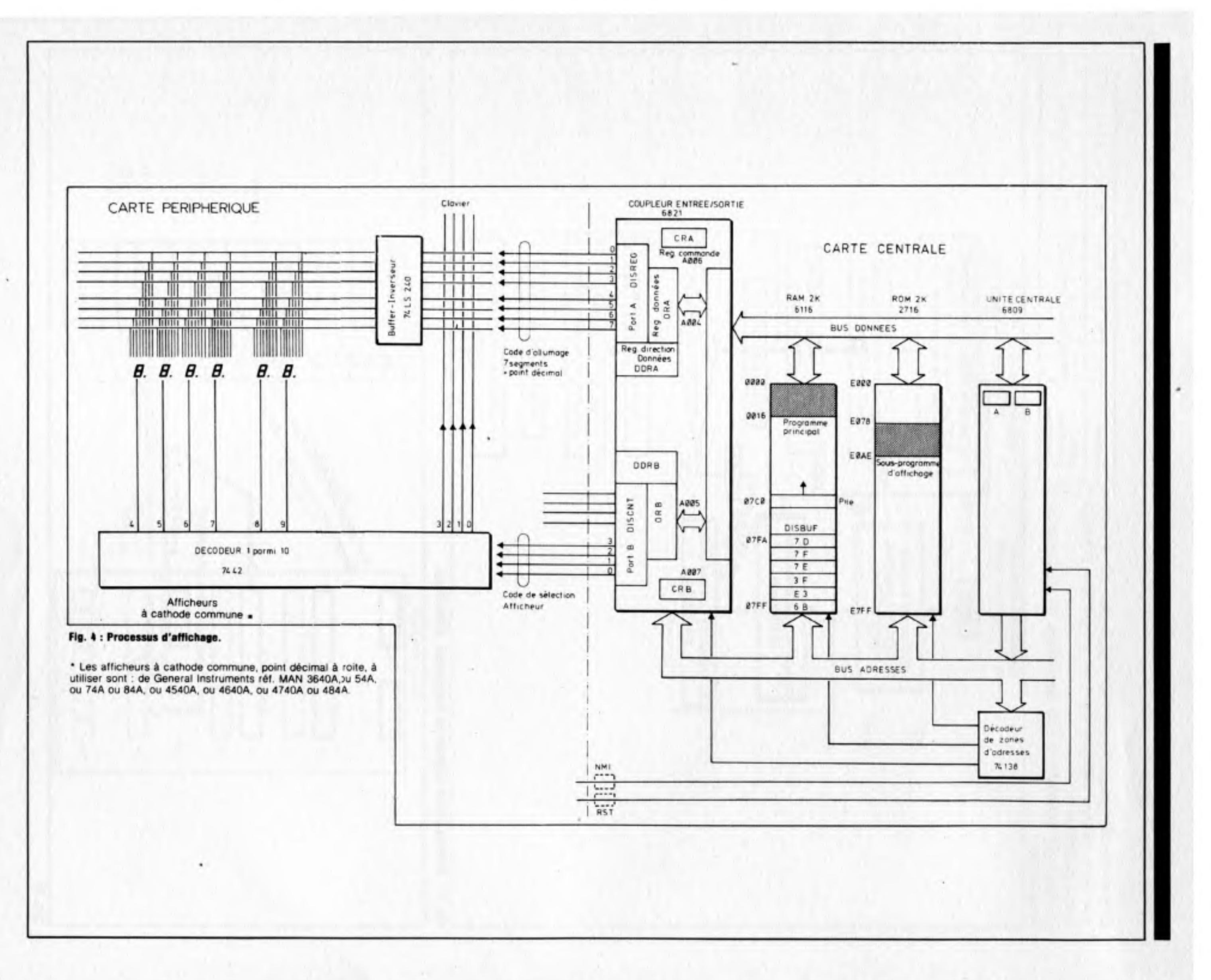


Fig. 4 : Processus d'affichage.

* Les afficheurs à cathode commune, point décimal à roite, à utiliser sont : de General Instruments réf. MAN 3640A,ou 54A, ou 74A ou 84A, ou 4540A, ou 4640A, ou 4740A ou 484A.

Listing des programmes-exercices

Programme n° 10 : Recopie de zone-mémoire

0 1 0 0	8 E 0 2 0 0			LDX	#$0200	X pointe l'adresse de début SOURCE
0 3	A 6 8 4		LOOP	→LDA	,X	Charger la donnée pointée
0 5	A 7 0 8			STA	8,X	La stocker à l'adresse (X) + 8
0 7	3 0 0 1			LEAX	1,X	Pointer adresse suivante
0 9	8 C 0 2 0 8			CMPX	#$0208	Fin de zone-mémoire ?
0 C	2 6 F 5			BNE	LOOP	Si non continuer rangement
0 E	3 F			SWI		Si oui arrêt et retour moniteur

Réponse « blanche » : utiliser l'instruction STA − 8,X

Programme n° 11 : Rangement d'octets

0 1 1 0	8 E 0 3 0 0			LDX	#$0300	X pointe en début de table
1 3	4 F			CLRA		Initialisation du générateur d'octets
1 4	5 F			CLRB		
1 5	E 7 8 B		LOOP	→STB	D,X	Contenu de B à l'adresse (D) + X
1 7	5 C			INCB		
1 8	2 6 F B			BNE	LOOP	Si B ≠ 00 continuer rangement
1 A	3 F			SWI		Si B = FF arrêt et retour moniteur

Réponse « rouge » : La valeur du déplacement étant codée en complément à 2, un déplacement de 256 adresses ($00 à $FF) nécessite un nombre d'au moins huit chiffres binaires significatifs et d'un bit de signe, soit neuf chiffres. D'où nécessité d'utiliser le registre D de capacité 16 chiffres.
Si l'on utilise un accumulateur A ou B, de huit chiffres, seuls les sept premiers chiffres (les bits nos 0 à 6) sont significatifs de la valeur absolue du « déplacement », le huitième chiffre (le bit n° 7) indiquant le signe. Or le contenu de l'accumulateur varie :
— de $00 (0000 0000 en binaire) à $7F (0111 1111), soit de + 0 à + 127 en décimal,
— puis de $80 (1000 0000 en binaire) à $FF (1111 1111) soit de − 128 à − 1.
On rangera donc bien des octets d'ordre croissant de $0500 à $057F ($0500 + 127) puis de $0480 ($0500 − 128 adresses) à $04FF ($0500 − 1).

Programme n° 12 : double décompteur

		Etiquette	Nbre de cycles			
0 1 2 0	8 E F F F F		3	→LDX	#$FFFF	Charge X avec 65 535
2 3	8 6 F F	LOOP1	2	→LDA	#$FF	Charge A avec 255
2 5	4 A	LOOP2	2	DECA		Décompter
2 6	2 6 F D		3	BNE	LOOP 2	Si A ≠ 0 continuer décompt. chif. infér.
2 8	3 0 8 2		4 + 2	LEAX	, − X	Sinon décompter chiffres supérieurs
2 A	2 6 F 7		3	BNE	LOOP 1	Si X ≠ 0 continuer décomptage
2 C	3 F		19	SWI		Sinon arrêt et retour moniteur

Le temps total d'exécution du programme est :

$$3 + ([(2 + 3) 255] + 6 + 3 + 2) \times 65\,535] + 19 = 84\,278\,032\ \mu s \simeq 84,3 \text{ secondes.}$$

Adresse-programme | Code-instruction | Etiquette | Langage Assembleur | Commentaires

```
00156                                 ****** ALLUMAGE DES AFFICHEURS ******

00158A E07B 34    16       A DISPRE PSHS   X,B,A
00159A E07D 8E    A004     A        LDX    #DISREG
00160A E080 4F                      CLRA
00161A E081 A7    02       A        STA    2,X      ACCES A DDRA
00162A E083 A7    03       A        STA    3,X      ACCES A DDRB
00163A E085 86    7F       A        LDA    #$7F
00164A E087 A7    84       A        STA    ,X       PA EN SORTIE
00165A E089 86    0F       A        LDA    #$0F
00166A E08B A7    01       A        STA    1,X      PB0-3 EN SORTIE
00167A E08D 86    04       A        LDA    #$04
00168A E08F A7    02       A        STA    2,X      ACCES A PA-DISREG
00169A E091 A7    03       A        STA    3,X      ACCES A PB-DISCNT
00170A E093 8E    07FA     A        LDX    #DISBUF
00171A E096 C6    03       A        LDB    #$03
00172A E098 5C                 RECOM INCB
00173A E099 C1    0A       A        CMPB   #$0A     TOUS AFFICHEURS SCRUTES?
00174A E09B 26    02  E09F          BNE    SCRUTA   NON,CONTINUER
00175A E09D 35    96       A        PULS   PC,X,B,A OUI,RETOUR SOUS GETKEY

00177                                 ****** ALLUMER UN AFFICHEUR APRES L'AUTRE ******

00179A E09F F7    A005     A SCRUTA STB    DISCNT   CHOISIR L'AFFICHEUR
00180A E0A2 A6    80       A        LDA    ,X+      PRENDRE CARACTERE DS
00181A E0A4 43                      COMA
00182A E0A5 B7    A004     A        STA    DISPEG   ALLUMER SEGMENTS
00183A E0A8 86    A0       A        LDA    #$A0              DISBUF
00184A E0AA 4A                 DLY1 DECA
00185A E0AB 26    FD  E0AA          BNE    DLY1     DUREE#1MS
00186A E0AD 20    E9  E098          BRA    RECOM    ALLUMER TS AFFICHEURS
```

Listing du sous-programme d'affichage.

Chapitre 3

Rôle des interruptions matérielles et logicielles

Ce troisième chapitre termine la première série consacrée à la présentation et à la réalisation de la maquette microkit 09 ainsi qu'à l'apprentissage des techniques de base de sa programmation. Il nous permettra de mieux comprendre et d'utiliser les interruptions du 6809 pour gérer des périphériques (PIA, ACIA, GDP, Timer, CAD, CDA, GPIA... *) ou pour démarrer un programme à partir d'une sollicitation extérieure.

Avant tout, nous sommes en droit de nous poser la question suivante : «qu'est-ce qu'une interruption ?».

C'est un moyen matériel (donc un signal représenté par le changement d'état d'une ligne) ou logiciel (donc une instruction placée dans le programme en cours d'exécution), qui permet :

- d'interrompre un programme en cours
- de traiter prioritairement un programme par rapport à un autre qui se trouve, par principe, moins prioritaire.
- de revenir, éventuellement, à la situation où l'on se trouvait avant la demande d'interruption ou d'attendre une autre interruption (dans ce cas, le CPU ne sera affecté qu'à une tâche de gestion d'interruption).

Le processeur peut ainsi traiter des problèmes «en temps réel»... limité seulement par sa vitesse de traitement en fonction des besoins extérieurs.

Le processeur 6809 possède un système très complet d'interruptions :

- interruptions logicielles qui viennent du programme lui-même (demande d'arrêt du programme, exécution du programme pas-à-pas pour une visualisation automatique des registres du microprocesseur, demande de lecture ou d'écriture sur un ou des organes périphériques...).

Les interruptions correspondantes s'appellent : «Software Interrupt».

SWI
SWI2
SWI3

- Interruptions matérielles qui sont au nombre de 3 :
  - NMI (Non Maskable Interrupt) : interruption non masquable
  - FIRQ (Fast Interrupt Request) : demande d'interruption rapide
  - IRQ (Interrupt Request) : demande d'interruption
- Mise en attente ou synchronisation sur un évènement extérieur dont la présence est signalée par une ou plusieurs entrées d'interruptions : ce sont les instructions.
  - CWAI (Clear and Wait Interrupt) : attente d'interruption... Nous verrons plus loin à quoi sert le Clear.
  - SYNC (attente d'une synchronisation externe).

Pour corser le tout, citons l'existence de 2 broches (fig. 1) Halt et Dma/Breq servant à déconnecter le microprocesseur de son environnement afin de permettre des traitements plus spécialisés.

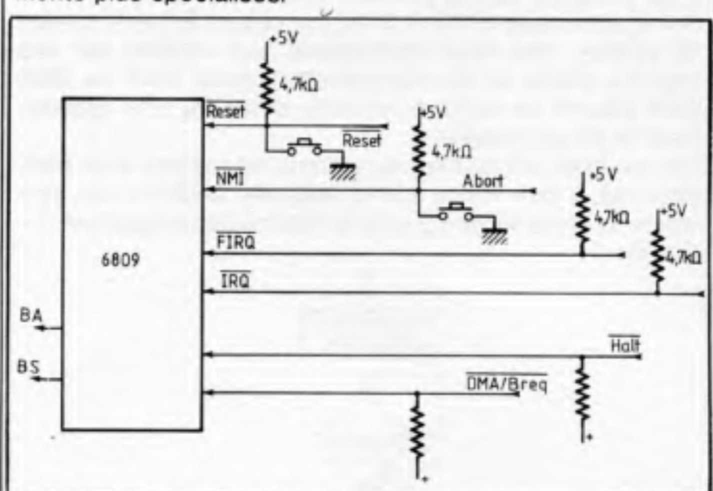


Fig. 1 : Les lignes d'interruptions matérielles.

Afin de nous dire où il en est de ses pérégrinations, le microprocesseur positionne deux lignes de sortie BA et BS selon 4 états possibles : (fig. 2).

BA= Bus disponible	BS= état du B	états du CPU
0	0	fonctionnement normal
0	1	Interruption quelconque reconnue
1	0	SYNC acceptée
1	1	Halt ou Bus accordé

BA = Bus Available
BS = Bus Status

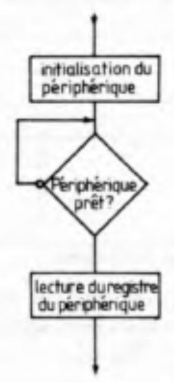
Fig. 2 : Table des états du CPU.

* ACIA : Asynchrone Communication Interface Adapter = Périphérique d'entrée-sortie série ; GPIA = General Purpose Interface Adapter = Interface d'applications générales (Bus IEEE, IEC) ; GDP = Graphic Display Processor = Processeur graphique.

Les lignes d'interruptions sont toujours actives à l'état bas et les entrées du CPU sont à collecteur ouvert, ce qui permet de relier plusieurs périphériques sur la même ligne constituant ainsi un «ou câblé».
Afin de permettre une bonne gestion sans confusion des interruptions, l'unité de séquencement du microprocesseur est programmée pour les exécuter en considérant leur priorité respective ; depuis la plus importante (celle qu'il faut exécuter avant toute autre) jusqu'à la plus faible (celle qu'il faut exécuter après toute autre).
Les priorités sont les suivantes :

$\overline{\text{RESET}}$
$\overline{\text{NMI}}$
SWI
$\overline{\text{FIRQ}}$ — Priorités décroissantes
$\overline{\text{IRQ}}$
SWI2 SWI3

Ainsi, si $\overline{\text{FIRQ}}$ et $\overline{\text{IRQ}}$ passent simultanément à l'état bas, c'est $\overline{\text{FIRQ}}$ qui sera la première prise en compte.
Remarquons que SWI2 et SWI3 possèdent le même niveau de priorité ; ces deux interruptions sont utilisées par des logiciels d'aide au développement (comme UNIX ou 0S9) pour assurer un «service request» et facilite ainsi grandement la programmation.
Par exemple, s'il faut lire le contenu du registre d'un périphérique, il faut écrire, par la méthode traditionnelle, des lignes de programme comme le montre l'organigramme ci-contre :



Par le «service request», il suffit tout simplement d'écrire un code qui pourrait être : 0S9 I$READ.
Le logiciel interprète ce code comme un SWI2 suivi d'une adresse qui lui permet de faire l'exécution demandée et de revenir au programme principal.
De toute façon, quelle que soit l'interruption demandée, le microprocesseur doit :
- interrompre le programme principal (ou en cours)
- garder tout ou partie du contexte dans une pile
- exécuter une séquence privilégiée, reflet du type de traitement d'interruption
- prendre en compte l'interruption toujours après l'exécution complète d'une instruction (sinon, «bonjour les dégâts !»).

Le microprocesseur doit savoir, à tout moment, où se brancher pour exécuter l'interruption demandée.
Il dispose ainsi de 14 adresses mémoires comprises entre $FFF2 et $FFFF qui lui permettront de savoir où se brancher pour exécuter l'interruption désirée ; en outre, la pile S sauvegarde tous les registres en mémoire, y compris le compteur de programme, pour permettre au CPU de revenir au programme qui était en cours d'exécution avant l'interruption.
Le programmeur doit être très vigilant sur la gestion de cette pile, la moindre erreur est souvent lourde de conséquences.
Les cases mémoires comprises entre $FFF2 et $FFFF sont affectées à des adresses représentatives des interruptions (fig. 3) hard et soft.
Ces adresses s'appellent des «vecteurs d'interruptions».
Ainsi, le vecteur d'interruption du Reset se trouve en $FFFE et $FFFF.

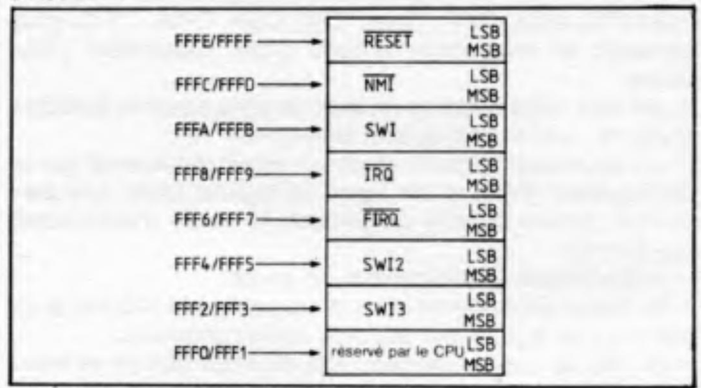


Fig. 3 : Table des vecteurs d'interruptions.

A cette adresse, le CPU va trouver une adresse qu'il placera dans son compteur de programme, ce qui lui permettra de se brancher au programme de reset demandé.
Ce qui revient à dire que le microprocesseur va chez Dupont (adresses $FFFE et $FFFF) demander l'adresse de Durant (le vecteur qui se trouve en $FFFE et $FFFF).
Il s'agit d'un adressage étendu indirect représenté par le mnémonique : JMP [$FFFE] codé par 6E 9F FF FE (fig. 4).
Aidez-vous des tableaux d'instructions présentés dans le

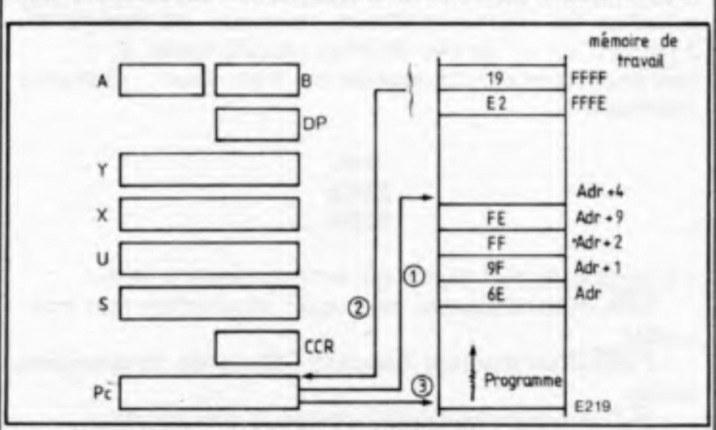


Fig. 4 : Séquencement des opérations de JMP [$FFFE].
① Le PC pointe sur Adr + 4 (instruction suivante).
② Chargement du vecteur E219 dans PC.
③ soit en E219 pour exécuter la routine demandée.

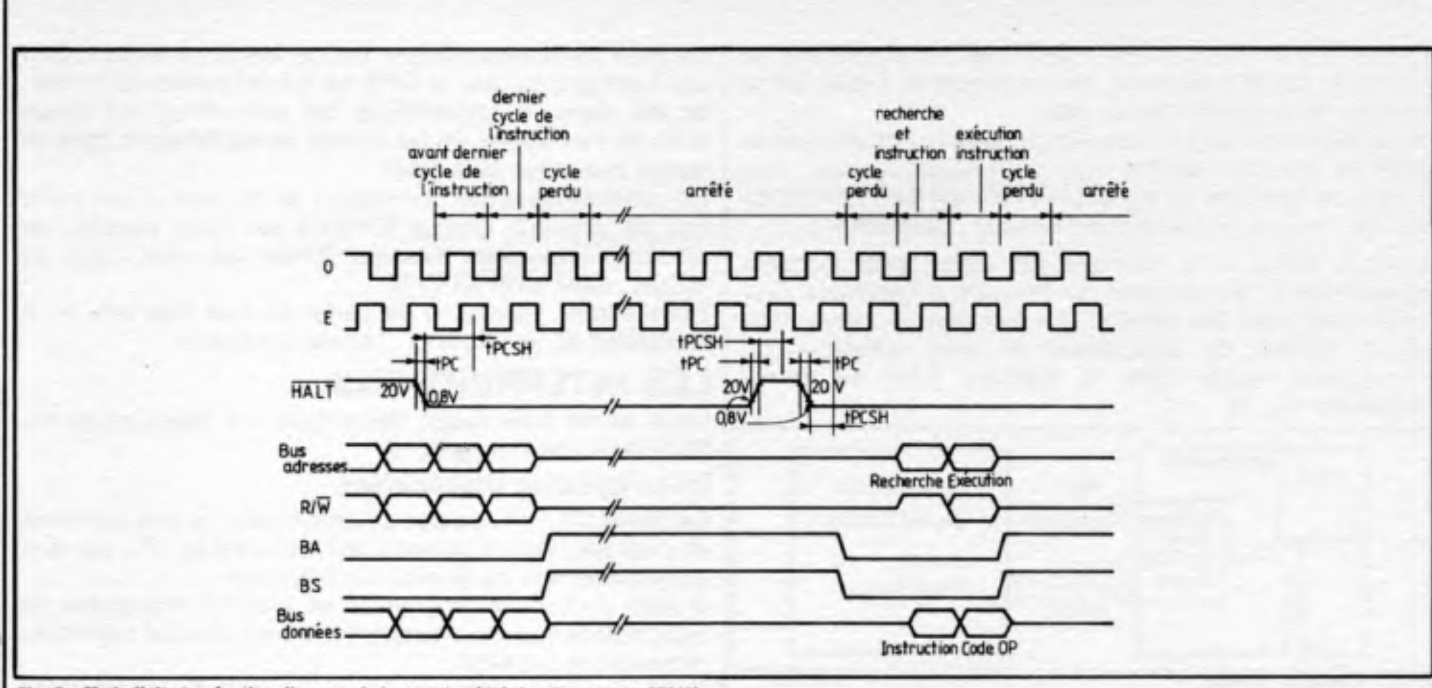


Fig. 5 : Mode Halt et exécution d'une seule instruction (tiré des documents-EFCIS).

chapitre deux pour retrouver le code opératoire.

Bien entendu, le microprocesseur effectue automatiquement ce branchement lorsqu'il rencontre une instruction d'interruption ou lorsqu'une de ses lignes hard passe à 0.

Revenons à l'explication des lignes Halt et Dma/Breq :

• Halt (fig. 5) : à l'état haut, le microprocesseur est en fonctionnement normal (BA = BS = 0), c'est le cas du microkit 09. A l'état bas, le microprocesseur s'arrête (BA = BS = 1) à la fin de l'instruction en cours et demeure à l'arrêt sans perte de données puisque l'horloge continue à fonctionner normalement pour rafraîchir les registres internes du CPU.

Le microprocesseur est donc en mode «off», ses bus d'adresses et de données ainsi que la ligne R/W étant à l'état haute impédance.

Un passage à 1, d'une durée d'un cycle, redémarre le CPU qui n'exécute qu'une seule instruction pour se mettre ensuite en mode off.

L'utilisation intelligente de cette ligne permet de travailler en mode multi-processeurs (fig. 6) : une unité centrale appelée «unité maître» assure l'aiguillage des tâches des unités asservies appelées «unités esclaves».

Les micro-ordinateurs modernes présentent souvent cette configuration : une unité centrale assurant la scrutation d'un clavier alphanumérique, une autre unité assurant l'affichage sur un moniteur vidéo, ..., l'unité maître assurant l'aiguillage et la gestion des tâches.

• Dma/Breq : Direct Memory Access/Bus Request (fig. 7) : cette ligne peut avoir deux utilisations différentes :

- accès direct à la mémoire
- rafraîchissement de mémoires dynamiques

Dans le premier cas, un circuit spécialisé appelé DMAC (Direct Memory Access Controler) fait une demande d'accès au bus en mettant à 0 la broche Dma/Breq du CPU ; ce dernier transfère le contrôle au DMAC en mettant ses lignes BA = BS = 1.

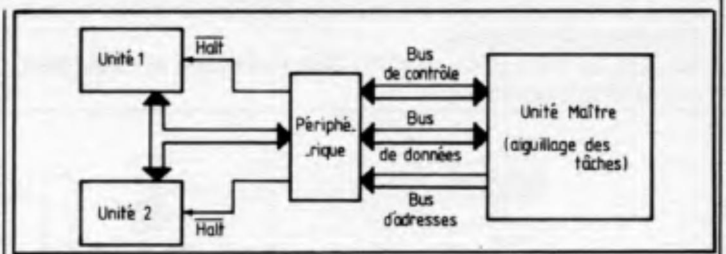


Fig. 6 : Configuration multi-processeurs.

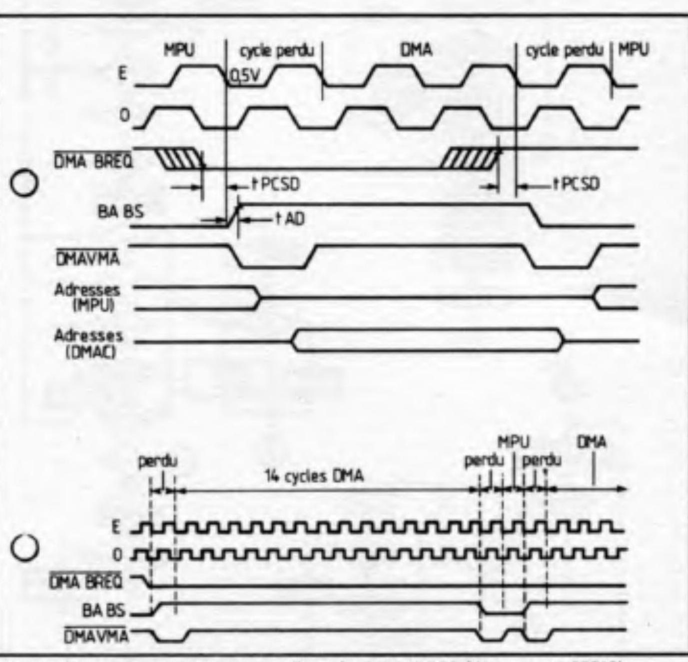


Fig. 7 : Diagramme des temps de l'entrée DMA/BREQ (documents FFCIS).
① Accès direct à la mémoire.
② Auto-rafraîchissement en DMA.

Durant tout le temps où BA = BS = 1, le bus d'adresses se trouve en haute impédance, ce qui permet au DMAC de se charger de la gestion de ce bus.

Il faut remarquer qu'un cycle est perdu lorsqu'on accède au DMA et lorsqu'on rend la main au microprocesseur, d'où l'utilité de fabriquer un signal $\overline{DMAVMA}$ qui tient compte de ce fait : on exécute seulement lorsque $\overline{DMAVMA}$ = 1.

L'accès direct à la mémoire est utilisé pour accéder rapidement à des données se trouvant à l'extérieur d'un micro-ordinateur (en général une mémoire de masse telle qu'un lecteur de disquettes) et pour effectuer un chargement rapide dans la mémoire RAM du micro-ordinateur (fig. 8).

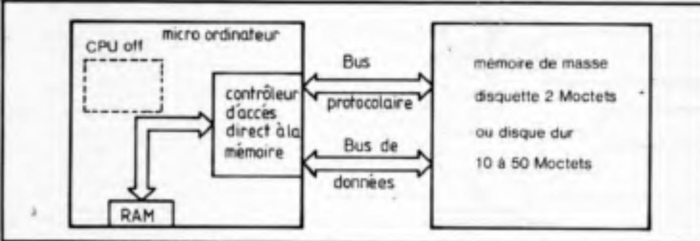


Fig. 8 : Processus de DMA.

Dans le deuxième cas, on arrête le CPU pour rafraîchir des mémoires dynamiques.

Au-delà de 14 cycles, le CPU doit reprendre la main pour auto-rafraîchissement.

Ce type de fonctionnement est de moins en moins utilisé car il oblige à bloquer le CPU, ce qui fait perdre du temps ; or, les mémoires dynamiques ont aujourd'hui des temps d'accès très courts, ce qui permet de les rafraîchir dans un temps beaucoup plus court.

On remarque que des données ne se trouvent jamais sur le bus de données lorsque E = 0 : il est donc possible de rafraîchir une case mémoire durant ce court laps de temps... sans arrêt du CPU.

Nous aurons l'occasion de parler de tout cela lors de la réalisation du microcomp... l'année prochaine.

LES INTERRUPTIONS

Nous allons maintenant décortiquer les interruptions du 6809 (flow chart de la fig. 9).

Interruptions matérielles

Le Reset (fig. 10) : il s'agit de l'interruption la plus prioritaire et c'est bien normal puisqu'il faut bien dire au CPU par quoi commencer lors de la mise sous tension.

D'autre part, si un programme se «plante» (impossible de reprendre la main) une action sur le bouton reset permet de reprendre le contrôle.

Enfin, le programme de reset permet de charger les registres à des valeurs spécifiques nécessaires pour un programme donné (par exemple : initialisation de la pile S = \$07C0) sauvegarde de valeurs constantes en RAM etc...

Rappelons le rôle des flags

E F X I X X X X

Flag de contexte (Entire Flag) 1 = contexte entier 0 = contexte partiel

Masque de FIRQ 1 = masquée 0 = autorisée

Masque de IRQ 1 = interrupt. masquée 0 = interrupt. autorisée

Fig. 9 : Organigramme du 6809.

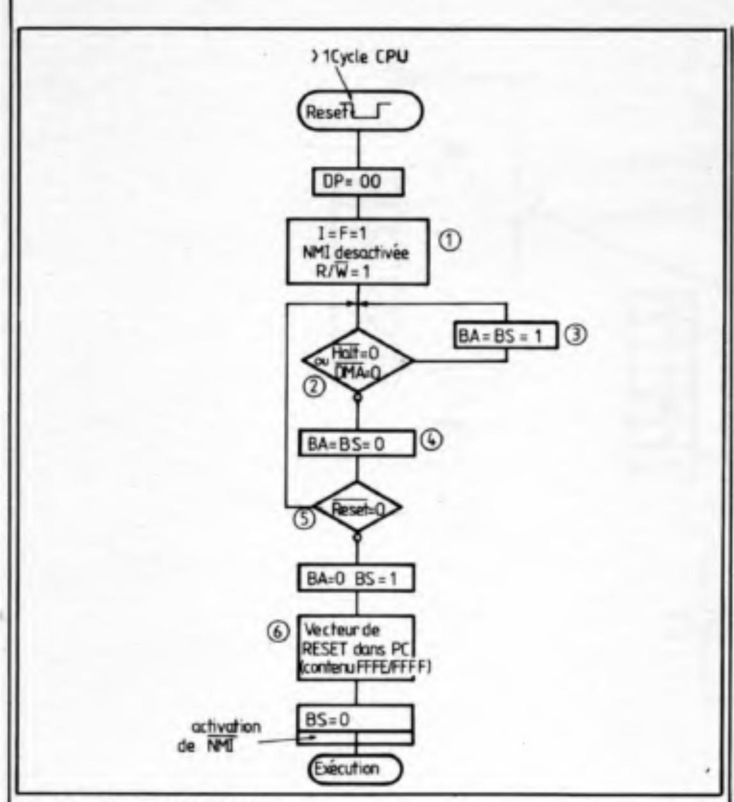


Fig. 10 : Séquence de Reset.

Cette séquence interne au CPU s'exécute en une dizaine de cycles.

Dès la mise sous tension, ou lors d'un appui sur la touche Reset, le CPU :

1. Charge le registre de page DP à 0 afin de se rendre compatible avec le 6800.
 - masque toute interruption telles que FIRQ, IRQ et NMI (inutile pour SWI, SWI2 et SWI3 puisqu'il s'agit d'instructions se trouvant dans un programme).
 - se met en lecture R/W = 1
2. Teste ses lignes Halt et DMA/Breq
3. Si 0 positionne les bus en haute impédance (BA = BS = 1) et attend un retour à la normale.
4. Dans ce cas, BA = BS = 0 sinon retour en 2.
5. Teste si Reset = 0, (en effet, une action sur la touche Reset dure sûrement plus longtemps qu'un cycle CPU = 1 µs..., nous sommes beaucoup moins rapides que le microprocesseur, on s'en doute !).
6. Dès que la touche est relâchée, on se met en mode reconnaissance d'interruption (BA = 0 et BS = 1) pour aller chercher le vecteur de Reset en FFFE/FFFF. Le contenu de ces deux cases est mis dans le compteur des programmes, puis le CPU se met en mode normal BA = BS = 0 pour exécuter le programme.

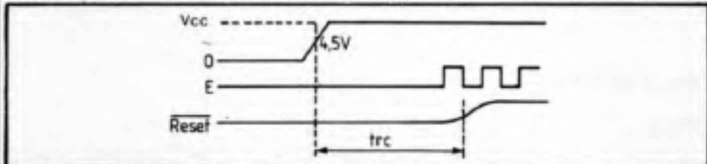


Fig. 11 : Auto Reset.

Notons qu'à la mise sous tension, l'horloge ne se met en route qu'au bout d'un temps trc ≃ 100 ms, le Reset n'est pris en compte qu'un cycle plus tard (fig. 11).

L'interruption NMI : (fig. 12)

Cette ligne d'interruption non masquable ne peut être ignorée (masquée) par le microprocesseur, elle est donc d'un niveau plus élevé que les autres interruptions (IRQ, FIRQ) mais moins élevée que le Reset qui la désactive. Si le microprocesseur est à l'arrêt, un front actif sur NMI sera mémorisé pour une réponse différée.

Puisque cette interruption est la plus prioritaire, on la réserve aux traitements devant résulter d'une défaillance d'alimentation (sauvegarde dans une mémoire C-MOS alimentée par batterie par exemple), ou pour visualiser le contenu des registres du CPU lorsqu'un programme «se plante» (bouton Abort sur le Microkit 09 et sur la plupart des systèmes de mise au point).

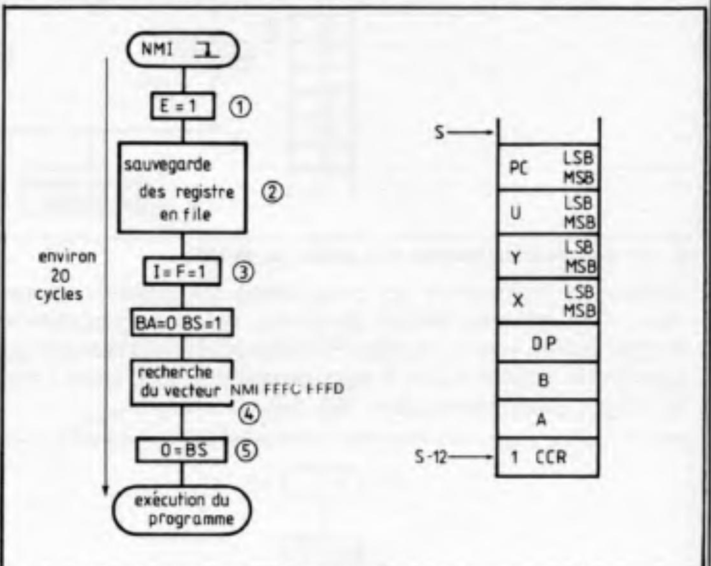


Fig. 12 : Organigramme du NMI.

Un front actif (négatif) sur l'entrée NMI du CPU provoque le déroulement de la séquence suivante :

① Le flag E du CCR se positionne à 1 pour indiquer que le microprocesseur sauve tous ses registres en Pile ②.

On retiendra que la Pile pointe toujours «au-dessus» de ce qu'elle va ranger et pointe toujours sur ce qu'elle vient de ranger.

Par exemple : avant sauvegarde S = $07C0
après sauvegarde S pointe sur le dernier registre rangé (CCR) en $07B4.

③ Le CPU masque les interruptions FIRQ et IRQ afin qu'elles ne soient pas exécutées durant le déroulement du programme de NMI. Puis il se met en mode reconnaissance de l'interruption et va chercher le vecteur qui se trouve en $FFFC/$FFFD pour le mettre dans le compteur de programme ④.

A ce stade, il se met en mode exécution BS = BA = 0 pour exécuter le programme de NMI (figure 13).

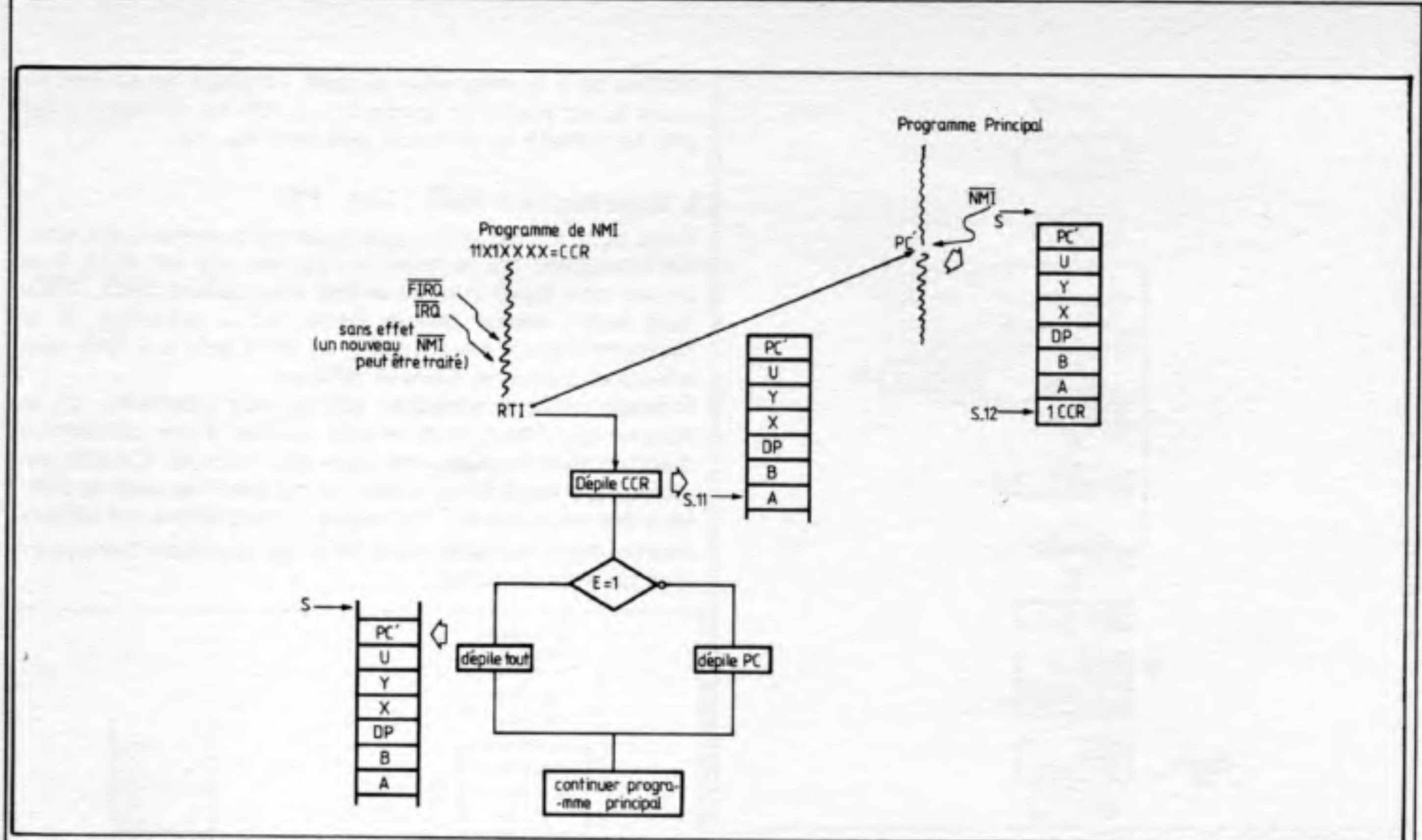


Fig. 13 : Exemple de déroulement d'un programme de NMI.

«Comment revient-on au programme principal ?» direz-vous. Ceci est bien simple, on insère, en fin de programme d'interruption, une instruction RTI (Return From Interrupt) qui permet de dépiler la pile S pour remettre le CPU dans l'état où il était avant interruption (fig. 14).

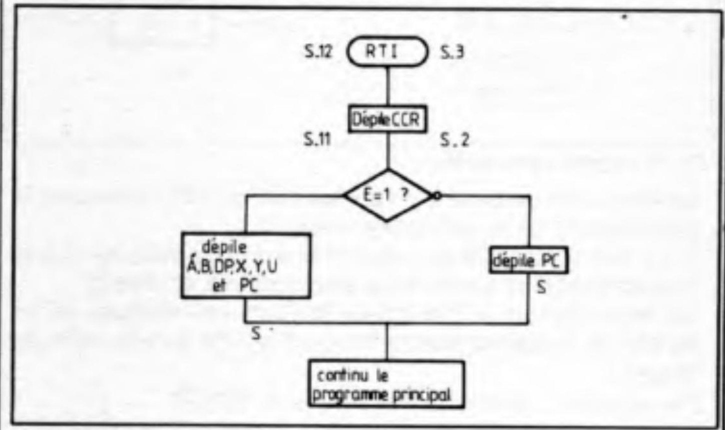


Fig. 14 : Organigramme de l'instruction RTI.

L'interruption $\overline{FIRQ}$ (fig. 15)

Cette interruption $\overline{FIRQ}$ est masquable par l'intermédiaire du bit 6 du CCR (flag F). Elle est plus prioritaire que $\overline{IRQ}$ puisque le CPU met un masque d'interruption sur le flag I du CCR.

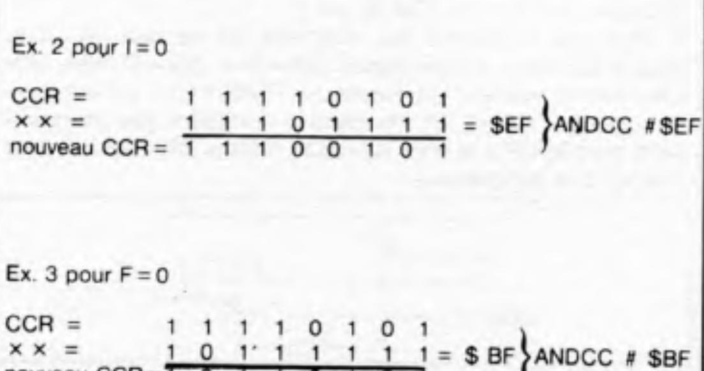
L'utilisateur peut sauvegarder dans la pile d'autres registres que PC et CCR grâce à l'instruction PSHS.

Signalons ou passage l'utilité des instructions ANDCC et ORCC qui s'utilisent exclusivement en adressage immédiat :

• ANDCC # $× × effectue le ET entre (CCR) et × × et place le résultat dans CCR.

Cette instruction positionne donc un bit particulier à 0

Ex. 1



Ex. 3 pour F = 0

CCR = 1 1 1 1 0 1 0 1
× × = 1 0 1 1 1 1 1 1 = $ BF } ANDCC # $BF
nouveau CCR = 1 0 1 1 0 1 0 1

FIRQ
F=1
ignorer l'interruption et continuer le programme en cours
environ 11 cycles
E = 0
sauve registres
S
PC
S-3
CCR
I= F = 1
BA=0 BS=1
recherche vecteur en $FFF6-FFF7
BS=0
exécution du programmeFIRQ

Exemple 1
ORCC #$50
FIRQ non traitée

Exemple 2
Programme de FIRQ
CCRQ
01X1XXXX
ANDCC # $BF
FIRQ traitée
Ignorées FIRQ IRQ
PSHS X,Y
(un NMI ne sera pas ignoré)
PC'
PC'
S-3
0 CCR
PULS X,Y
RTI

Fig. 15 : L'interruption FIRQ.

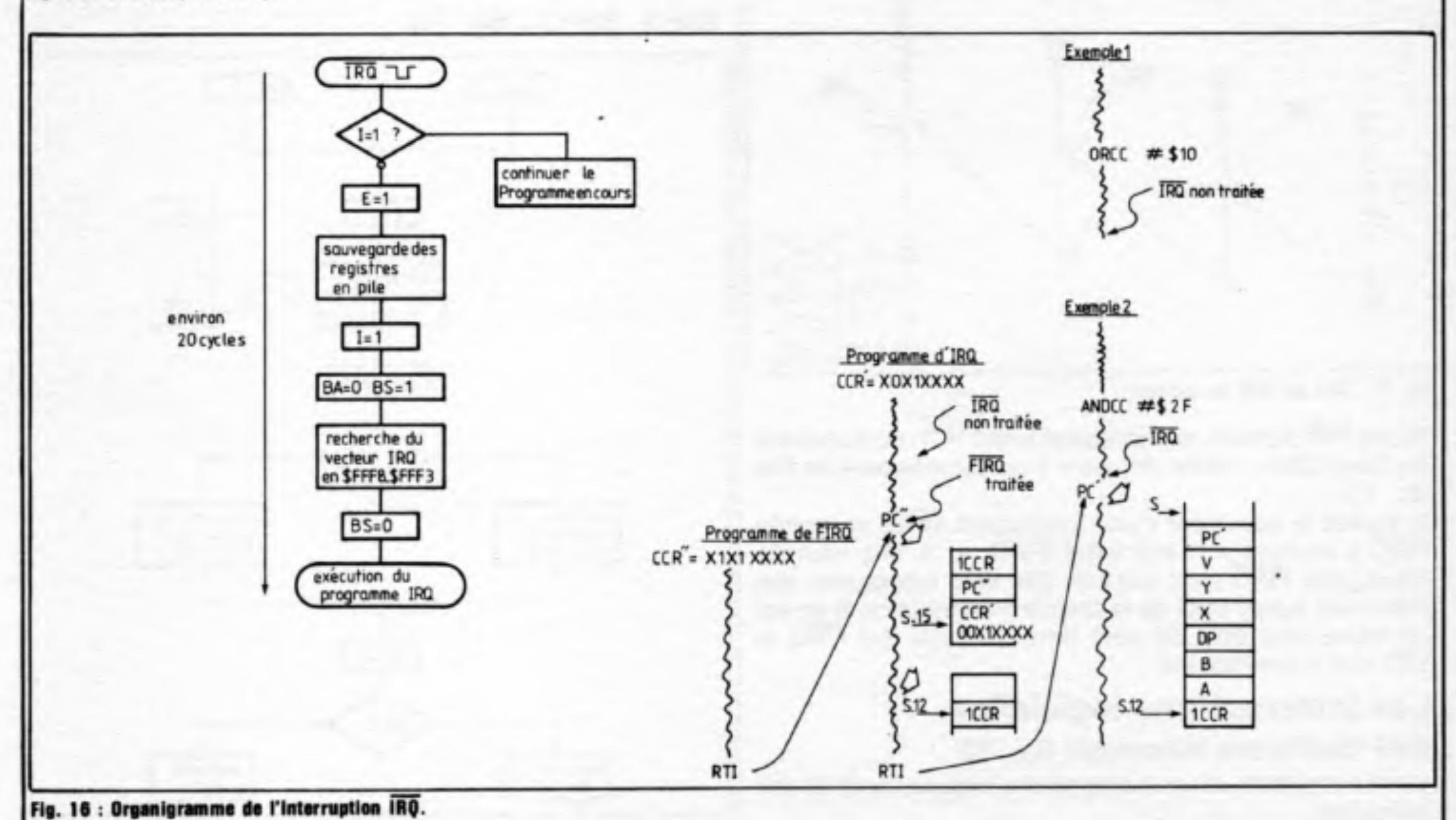


Fig. 16 : Organigramme de l'interruption IRQ.

• ORCC # $× × effectue le OU logique entre (CCR) et × × et place le résultat dans CCR.
Cette instruction positionne donc un bit particulier à 1.

Ex. 1 pour I = 1

	E	F	H	I	N	Z	V	C	
CCR =	0	1	1	0	1	1	0	1	flag à mettre à 1
× × =	0	0	0	1	0	0	0	0	= $ 10
nouveau CCR =	0	1	1	1	1	1	0	1	← recopié

recopié — imposé à 1 par le OU logique

Ex. 2 pour F = 1

	E	F	H	I	N	Z	V	C	
CCR =	0	0	0	0	1	0	1	0	
× × =	0	1	0	0	0	0	0	0	= $ 40 } ORCC # $40
nouveau CCR =	0	1	0	0	1	0	1	0	

L'interruption IRQ (fig. 16)

C'est l'interruption matérielle la moins prioritaire car le flag F n'est pas positionné à 1, on peut la masquer par l'intermédiaire du bit 4 du CCR (flag I)

Remarques sur les interruptions hard

1. Il faut éviter d'envoyer une seconde interruption $\overline{NMI}$ avant la fin du traitement de la première car $\overline{NMI}$ est toujours prise en compte !

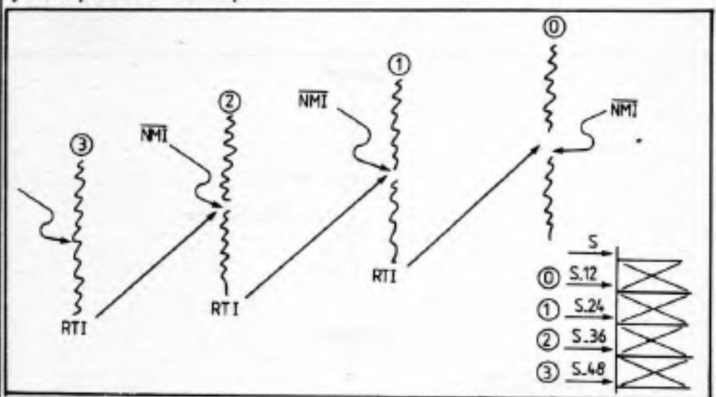


Fig. 17 : Gare aux NMI en cascade !

Si une $\overline{NMI}$ survient régulièrement avant la fin du traitement de l'interruption initiale, on arrive à un débordement de Pile (fig. 17).

2. Durant le traitement d'une interruption $\overline{FIRQ}$, si l'entrée $\overline{FIRQ}$ a retrouvé son état initial ($\overline{FIRQ}$ = 1), une nouvelle interruption $\overline{FIRQ}$ peut survenir. Elle sera mémorisée, son traitement suivra celui de la première interruption. Il en est de même pour $\overline{IRQ}$. On peut donc conclure que $\overline{FIRQ}$ et $\overline{IRQ}$ sont mémorisables.

Les interruptions logicielles

SWI (SoftWare Interrupt) fig. 18

Cette instruction, dans un programme, impose l'arrêt de son exécution.

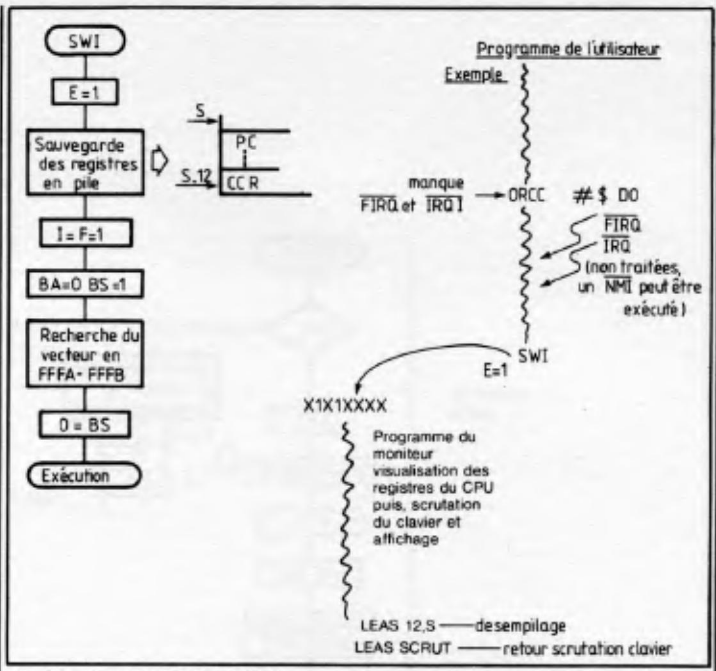


Fig. 18 : Organigramme de SWI.

Cette interruption est en générale réservée aux logiciels systèmes (par exemple arrêt d'un programme pour une visualisation automatique des registres du CPU). On remarquera que SWI est plus prioritaire que $\overline{FIRQ}$ et $\overline{IRQ}$ car son traitement entraîne le masquage de celle-ci.

SWI2, SWI3 : fig. 19

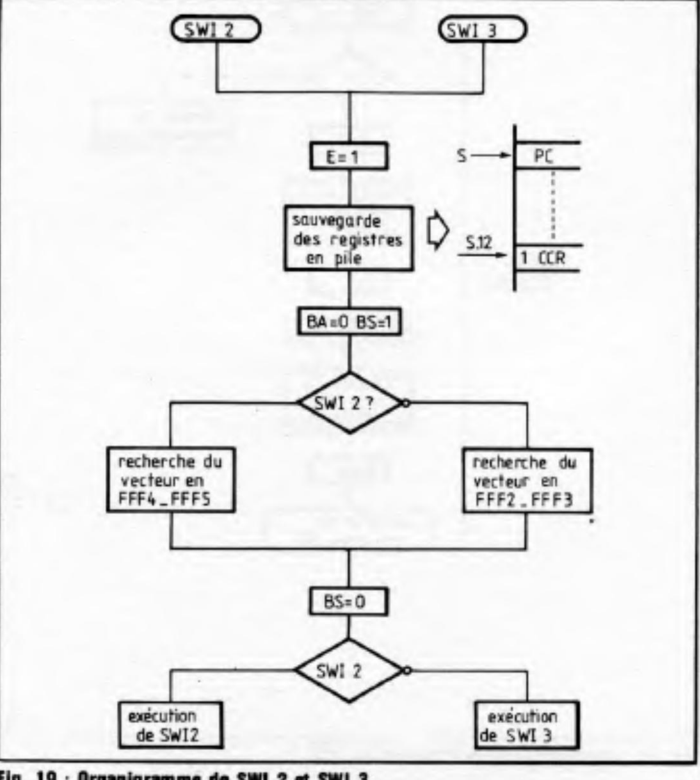


Fig. 19 : Organigramme de SWI 2 et SWI 3.

Ces deux interruptions ont un fonctionnement identique, elles peuvent être interrompues par toutes les autres interruptions du CPU.
Leur fonctionnement est identique à celui de SWI, seuls les masques d'interruption ne sont pas positionnés.

Les instructions d'interruptions

CWAI (Clear and Wait Interrupt : attente d'interruption) fig. 20 :
Cette instruction, qui occupe deux octets, joue deux rôles :
(1) Elle effectue le ET logique entre le contenu du CCR et une valeur immédiate. Le but à atteindre étant le même que l'instruction ANDCC # $ × × : mettre à 0 un flag particulier (en l'occurrence I et/ou F) d'où permission d'une interruption IRQ et/ou FIRQ.
(2) Arrêter le CPU qui ne démarrera que lorsqu'une interruption viendra.
Cette attente d'interruption met le CPU en veille mais non en haute impédance, BA et BS restent à zéro durant l'attente. Les registres internes sont toujours rafraîchis par l'horloge du CPU.

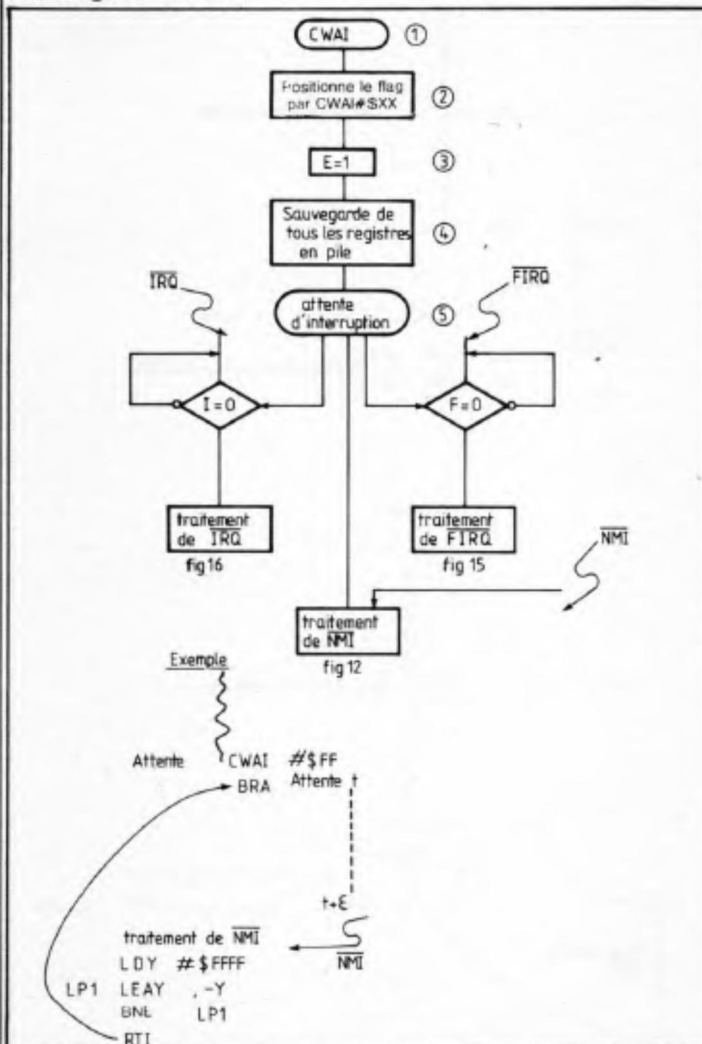


Fig. 20 : Organigramme de CWAI # $XX.

(1) CWAI arrête l'exécution du programme
(2) Le CPU valide ou masque les interruptions
CCR = FF → IRQ et FIRQ masquées
EF ← IRQ autorisée
BF ← FIRQ autorisée
AF ← IRQ et FIRQ autorisées
(3) E = 1 indique la sauvegarde totale du contexte du CPU
(4) Tous les registres internes sauf S sont sauvés dans la pile système.
(5) Le CPU se met en attente d'une interruption IRQ, FIRQ, (suivant le contenu du CCR) ou NMI.

Remarque : Lorsqu'une interruption survient, aucun autre état du CPU n'est sauvegardé avant la vectorisation du sous-programme de traitement de l'interruption. On peut donc utiliser l'interruption FIRQ avec une sauvegarde totale du contexte du microprocesseur (il s'agit d'un cas particulier à retenir).

SYNC (SYNchronisation) fig. 21 :
Il s'agit d'une instruction très puissante qui ne propose pas moins de 8 possibilités différentes !
Elle permet de synchroniser le déroulement du programme sur un évènement extérieur grâce aux lignes d'interruptions.

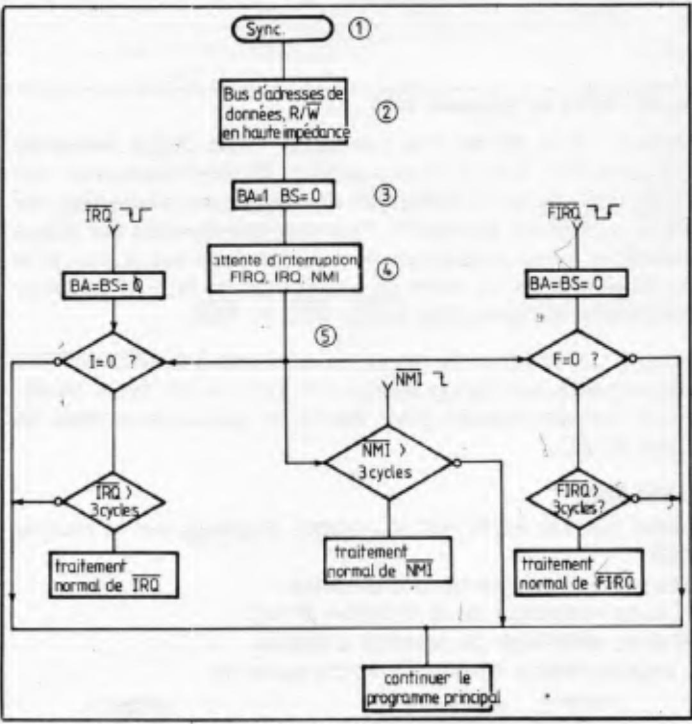


Fig. 21 : Organigramme de l'instruction SYNC.

(1) L'instruction SYNC arrête le CPU
(2) Les bus de données, d'adresses et la ligne R/W se mettent en haute impédance
(3) Le CPU indique qu'il est en attente de synchronisation
(4) Il reste dans cet état tant qu'il n'a pas reçu d'interruption NMI, FIRQ, IRQ

(5) Si une interruption se présente, sur $\overline{IRQ}$ par exemple (fig. 22), le fonctionnement redevient normal (BA = BS = 0).
Si $\overline{IRQ}$ est valide, et si le niveau bas dure plus de 3 cycles CPU, celui-ci exécute le traitement approprié
Au retour d'interruption, le processeur reprend le déroulement normal du programme.
Si l'interruption est masquée ou si le signal dure moins de 3 cycles CPU, le processeur continue le programme principal sans traiter l'interruption (le cas est le même pour $\overline{FIRQ}$).
Pour $\overline{NMI}$, il n'est pas nécessaire de tester son flag puisqu'il n'y en a pas.

Remarque : puisqu'il est possible de mettre le CPU en haute impédance, il est facile de conclure que l'instruction SYNC peut être utilisée pour assurer des synchronisations rapides avec des périphériques, cette méthode permet éventuellement d'éviter l'utilisation d'un circuit d'accès direct à la mémoire (DMA).

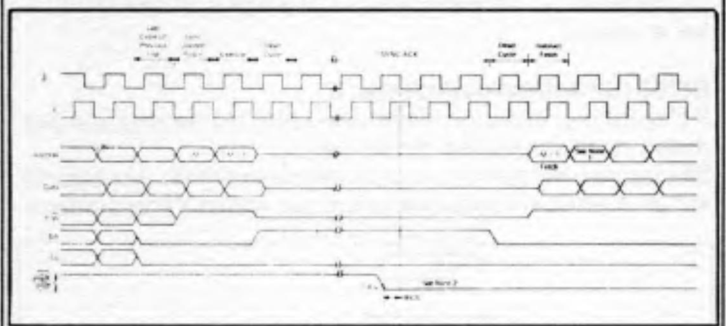


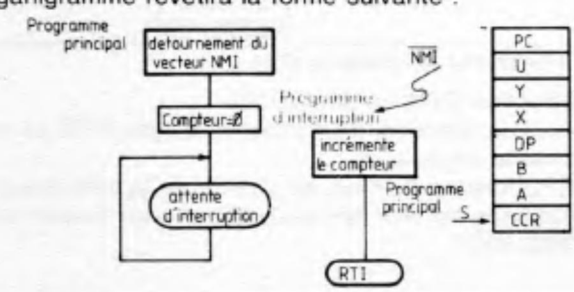
Fig. 23 : Timing de l'instruction SYNC.

Note 1 : Si le bit de masque est à 1 lors d'une demande d'interruption, le traitement continue. Si une interruption non masquable ou une interruption non masquée provoquée par $\overline{FIRQ}$ ou $\overline{IRQ}$ est acceptée, l'adresse positionnée sur le bus depuis le cycle précédent (M + 1) demeure sur le bus et le traitement continue avec ce cycle comme (M + 1) du chronogramme d'interruption $\overline{FIRQ}$, $\overline{IRQ}$ ou $\overline{NMI}$.

Note 2 : Si les bits de masques sont mis à 0, IRQ et FIRQ doivent être maintenus à l'état bas bien qu'un cycle seulement soit nécessaire pour mettre le processeur hors de l'état SYNC.

Exercice

Notre but est d'afficher le nombre d'appuis sur la touche NMI.
Nous allons utiliser deux méthodes :
1) avec utilisation de la fonction SYNC
2) avec affichage du nombre d'appuis
L'organigramme revêtira la forme suivante :



Exemple 1

```
10    LDA   ,X+
100   ORCC  #$DØ
105   Sync
      E
      IRQ   < 3 cycles
            ou > 3 cycles
106   PSHS  B,A   ] le programme redémarre sur IRQ
107   BRA   LP1   ]
```

Exemple 2

```
10    LDA    ,X+
100   ANDCC  #$EF ← I autorisée
105   Sync
      IRQ
      E
      IRQ   < 3cycles
106   PSHS  B,A   ] le programme redémarre sur IRQ
107   BRA   LP1   ]
```

Exemple 2

```
      LDA    ,X+
      ANDCC  #$EF ← I autorisée
      LP1 Sync
      IRQ    BRA LP1
      E
      IRQ   > 3cycles
Programme de IRQ
RTI
```

Fig. 22 : Différentes séquences d'exécution de SYNC suivant l'état de I et la durée de INA = 0.

1) Avec l'utilisation de la fonction SYNC.

Le programme principal sera logé en $0000, tandis que celui d'interruption sera logé en $0100.

Le détournement du vecteur NMI est simple à réaliser, celui-ci se trouve en $07DD (voir listing du moniteur). En fait, lorsqu'une interruption NMI est reconnue par le processeur, celui-ci va chercher une adresse en $FFFC/$FFFD.

On y trouve $E 7EA, cette valeur est mise dans le compteur de programme et le CPU va exécuter le programme se trouvant à cette adresse, on y a :

E7EA LDX > SAVNMI B7 07DD charge dans X le contenu se trouvant à l'adresse 07DD-07DE

JMP ,X saut à l'adresse pointée par X ($ E272)

En 07DD, on est dans la mémoire RAM, ce qui permet de lire et d'écrire (donc de détourner le vecteur qui se trouve là), il suffit donc dans le cas de notre exemple, de mettre $0100 aux adresses $07DD-07DE.

Le compteur peut être l'accumulateur A (ou une adresse mémoire)

Le programme sera donc le suivant :

```
Programme principal :           $0000  LDX    #$0100     8E  01  00
                                       STX    >SAVNMI    BF  07  DD
                                       CLRA              4F
                                Retour SYNC              13
                                       BRA    Retour     20  FD
Programme d'interruption :      $0100  INC    1,5        6C  61
                                       RTI               3B
```

Un appui sur la touche NMI aura pour conséquence de se dérouter sur la routine d'interruption qui se trouve en $0100, on incrémente l'accumulateur A puis, un RTI nous ramène à l'adresse $0008.

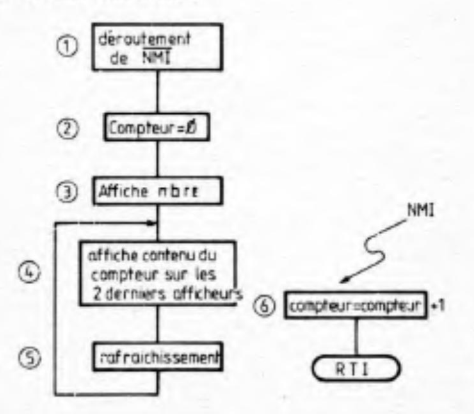
Un branchement relatif positionne le CPU en haute impédance (SYNC) pour lui permettre d'attendre une nouvelle interruption.

Un appui sur Reset, réinitialise le moniteur pour nous permettre de «reprendre la main» puis r A nous permettra de vérifier que le comptage s'est bien effectué.

2) Avec affichage du nombre d'appuis :

Le programme sera dans ce cas un peu plus complexe. Puisqu'il s'agit d'afficher, le CPU devra constamment travailler, donc les instructions CWAI ou SYNC seront interdites.

L'organigramme est le suivAnt :



Les parties 1, 2 et 6 sont identiques dans leur principe à l'exemple précédent, nous laisserons donc les instructions correspondantes en place.

La partie 3 fait appel à l'écriture de nbrE dans disbuf, on se servira de Y pour pointer sur disbuf et X contiendra la valeur à afficher : nbrE se traduit par $45754179.

La partie 4 consiste à convertir une valeur hexadécimale se trouvant dans l'accumulateur A en une valeur d'affichage 7 segments, les sous-programmes L7SEG ($ E0FC) et R7SEG ($ E100) se chargeront de cette conversion (rappelons que la valeur à convertir doit obligatoirement se trouver dans A, il se trouve détérioré en fin de sous-programme ; ce qui nous amènera à utiliser B comme compteur).

La partie 5 sert à balayer les afficheurs de manière à visualiser l'affichage, le sous-programme DISPRE ($ E07B) se chargera de cela.

D'où le programme suivant :

```
$0000    LDX #$0100
         STX > SAVNMI     Détournement de NMI
         CLRB             Compteur = 0
         LDY # $07FA      Pointe sur disbuf (1er aff.)
         LDX#$4575
         STX,Y + +        Ecrit nbrE dans disbuf et pointe
         LDX#$4179        en disbuf +4=4e afficheur
         STX,Y + +
Retour   TFR B,A          Compteur dans A
         PSHS A           Sauve compteur à convertir en
                          7 segments
         LBSR L7SEG       Conversion des 4 bits de poids
                          forts
         STA,Y +          Stocke dans 5e afficheur
         PULS A           Reprend compteur
         LSBR R7SEG       Conversion des 4 bits
                          de poids faibles
         STA,Y            Stocke dans 6e afficheur
         LEAY,-Y          Revient au 5e afficheur
         LBSR DISPRE      Allume les afficheurs

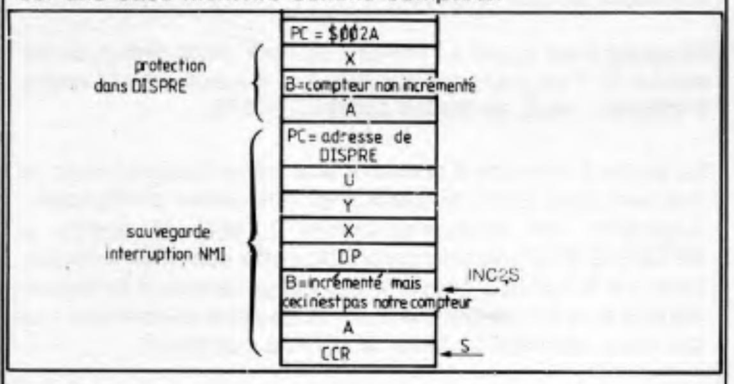
002A     BRA Retour       Recommence
                          NMI
$0100    Inc 2,S          Incrémente B
         RTI
```

Si vous essayez ce programme, vous constaterez qu'il ne fonctionne pas à tous les coups. On constate en effet que l'on effectue une sauvegarde de B (notre compteur) en Pile dans le sous-programme DISPRE.

Or, il y a de fortes chances pour que l'interruption se produise dans ce sous-programme puisque c'est lui qui réclame le plus de temps pour s'exécuter.

En conséquence, le compteur que l'on incrémente dans le programme d'interruption n'est pas notre compteur (voir dessin).

Il est donc exclut de prévoir A, B, X comme compteur, il nous reste Y et U mais la solution la plus simple étant d'utiliser une case mémoire comme compteur.




```
$0000          LDX # $0100       8E 0100
               STX > SAVNMI      BF 07DD
               CLR > CASE        7F 00 30
               LDY # $07FA       108E 07FA
               LDX # $4575       8E 4575
               STX, Y + +        AF A1
               LDX # $4179       86 41 79
               STX, Y + +        AF A1

$ 00 17 retour LDA > CASE        B6 00 30
               PSHS A            34 02

$ 001C         LBSR L7SEG        17 (E0FC) E0DD
               STA, Y +          A7 A0
               PULS A            35 02

$ 0023         LBR R7SEG         17 (E100) EDOA
               STA, Y            A7 A4
               LEAY, - Y         31 A2
               LBSR DISPRE       17 (E07B) E04E
               BRA retour        20 ($0017) E8
               CASE EQU $0030

$ 0100         INC > CASE        7C 00 30
               RTI               3B
```

Cet exemple montre bien qu'il faut être très prudent dans la gestion de la pile lors d'interruptions ; il est quelquefois préférable de se réserver une adresse buffer pour y faire un travail particulier (ici notre compteur) afin de ne pas subir une perte de données à cause d'une mauvaise gestion de la pile.

Chapitre IV

Aspects du Logiciel

Le logiciel présenté ici est celui du Microkit 09. Celui du MOPET, intitulé Micromon-Plus reste très semblable dans toutes ses formes et sera présenté dans le tome 2.

Présentation Générale

Le logiciel est constitué par le programme moniteur NANO-MON REV 1.8 implanté en EPROM 2716 2 K × 8 bits depuis l'adresse E000 à l'adresse E7FF. Le programme principal s'articule autour du programme RESET comme le montre l'organigramme ci-dessous. A la mise sous tension, ou dès l'appui sur la touche «Reset» le signe « – » est visualisé sur l'afficheur de gauche. Dès lors, seules les touches M. BP.R.CN.L.P.GO. sont influentes. L'appui sur l'une de ces touches provoque l'exécution de l'un des sous-programmes EXMEMO, BPOINT, FONREG, etc... aux adresses précisées sur l'organigramme général. Ainsi, la touche M permet l'examen et le changement du contenu des mémoires. L'emplacement mémoire visé doit être précisé par son adresse héxadécimale, entrée par l'intermédiaire du clavier et contrôlée par le sous-programme BADD. La donnée correspondante est alors affichée. Cette donnée peut être changée.

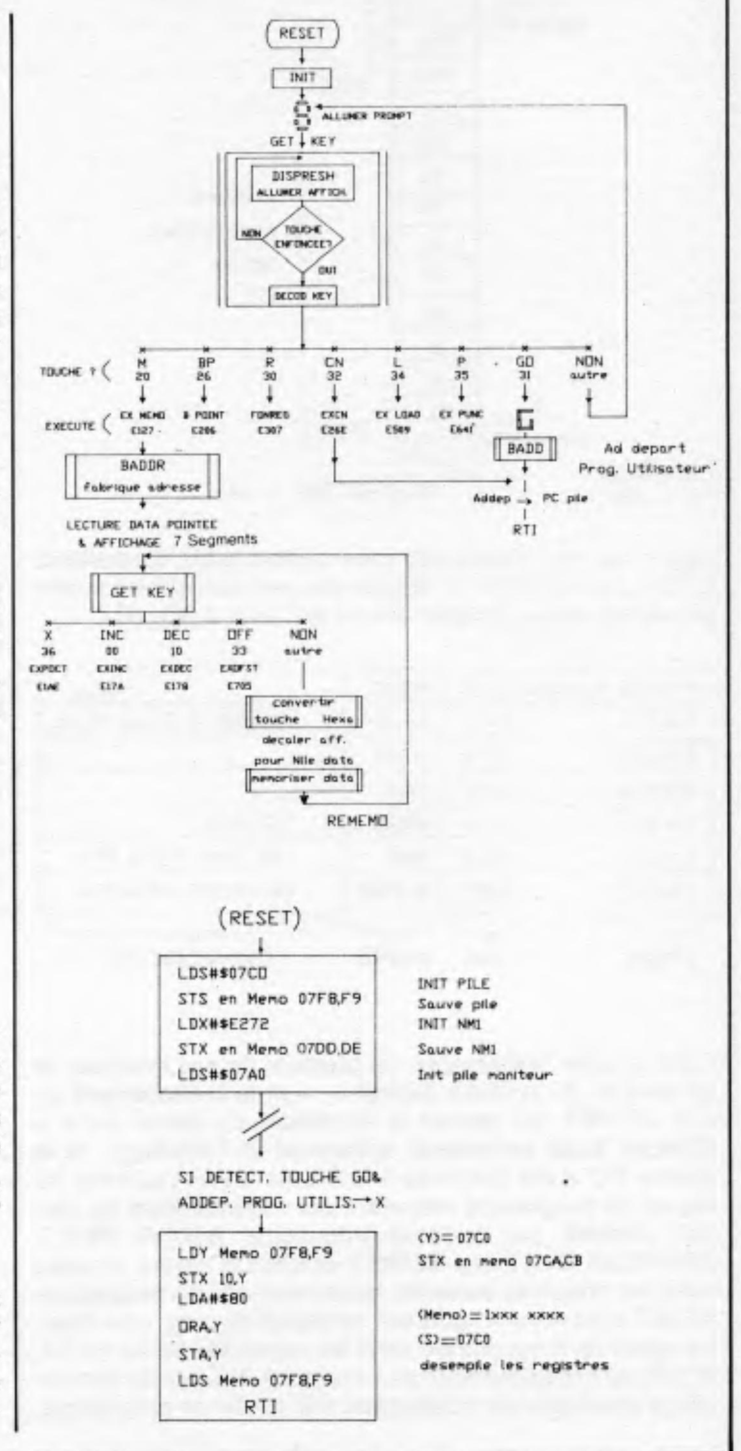
Les caractéristiques essentielles du programme moniteur sont décrites dans les paragraphes suivants. Nous analyserons les programmes et sous-programmes suivants :

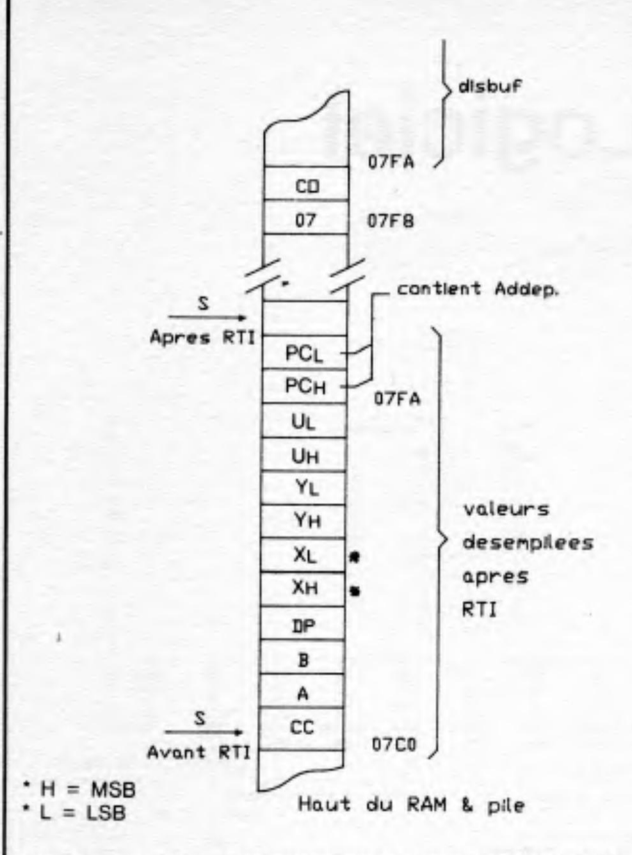
- Mise en route RESET
- Clavier et affichage
- Examen et changement du contenu des mémoires
- Visu et changement du contenu des registres
- Calcul automatique d'offset
- Interface cassette

Mise en route & Initialisations

Un appui sur la touche «RESET» force le μp à lire les emplacements mémoire FFFE, FFFF. Le décodage des adresses hautes ($A_{15, 14, 13}$) du système étant partiel par tranches de 4 Koctets (1000 Hexa) et par tranches de 2 Koctets (800 Hexa) pour les adresses basses de la RAM et de l'Eprom, le μp prend en compte les adresses images E7FE, E7FF. Il y trouve E219 et exécute le programme «RESET» partant de cette adresse.

Ce programme initialise le pointeur de pile du système à la valeur 07C0 et le vecteur NMI à la valeur E272. Le pointeur d'interruption est mis en RAM à l'Adresse 07DD,DE. L'utilisateur peut changer cette valeur et obliger le programme d'interruption à exécuter un programme spécifique d'inter-





ruption au lieu d'exécuter celui commençant à l'adresse E272. La composition de la table des vecteurs d'interruption permet de mieux comprendre ce qui vient d'être dit.

E7F2,F3 contient	077A	RSWI3	
E7F4,F5	E27C	RSWI2	ASSOCIE A Break Point
E7F6,F7	0775	RFIRQ	
E7F8,F9	0770	RIRQ	
E7FA,FB	027A	RSWI	VISU.REG
E7FC,FD	E7EA	RNMI	LDX Memo 07DD&JMP,X
E7FE,FF	E219	RESTAR	PROGRAMME PRINCIPAL
EPROM	RAM	INTERR.	PROGR.EXECUTE

C'est ensuite l'initialisation du pointeur de pile moniteur, la génération du symbole prompt « – » et le branchement au s/p GETKEY qui permet la scrutation du clavier (pour y détecter toute commande actionnée) et l'affichage. Si la touche GO a été enfoncée il faut alors entrer l'adresse de départ du programme utilisateur par l'intermédiaire du clavier contrôlé par le sous-programme BADDR (BUILD ADDRESS). Ces 2 s/p GETKEY et BADDR seront détaillés dans les chapitres suivants. Nous revenons au programme RESET avec dans X l'adresse de départ du prog. utilisateur. La valeur de X est placée dans les cases Mémoires (07CA, 07CB) qui correspondront au contenu du PC lors du désempilage provoqué par l'instruction RTI de fin de programme.

C'est ensuite la mise à 1 du bit 7 de la case Mémoire 07CC. C'est aussi la mise à 1 du flag E du registre CC vis-à-vis du désempilage. C'est enfin la réinitialisation du pointeur de pile du système et le RTI. Cette dernière instruction provoque le désempilage de tous les registres et le PC se trouve chargé à l'adresse départ prog. utilisateur. Le système exécute alors le prog. utilisateur.

NMI - Un appui sur la touche NMI provoque le lancement du programme d'interruption à l'adresse contenue en 07DD,DE. Si ces cases mémoires n'ont pas été forcées par l'utilisateur, elles contiennent les valeurs E272, valeur injectée par le programme RESET.

ROUNMI charge A avec le MSB de l'adresse en cours de programme (LDA 10,S ; c.-a-d. MSP du PC dans A). Si le poids fort d'adresse est Eα EQ, ROUNMI provoque un branchement à R POINT. Sous contrôle moniteur (Adresse > E000), la touche NMI a même action que la touche RESET. Si un programme utilisateur est lancé (Adresse en cours inférieure à E000 et même à 07FF dans le cas de notre système), ROUNMI provoque 1 branchement à RSWI (VISU des Registres).

Initialisation de la fonction NMI

```
E272 A6   6A          A ROUNMI LDA   10,S
0274 84   FO          A        ANDA  #$FO
E276 81   EO          A        CMPA  #$EO
E278 27   AD   E227            BEQ   RPOINT
E27A 20   2B   E2A7            BRA   RSWI

E7EA BE   07DD        A RNMI   LDX   SAVNMI
E7ED 6E   84          A        JMP   ,X
```

RSWI2 - est associé à la fonction BP (Break Point) de la façon suivante : lors de la mise d'un point d'arrêt, l'instruction utile (sauvegardée en Memo 07DF, 07FO) est remplacée automatiquement par 10 3F = instruction SWI2 (voir BPOINT). Le programme utilisateur s'arrête sur cette interruption, va lire le cases Memo E7F4,F5 pour y trouver E27C. Il exécute alors le programme RSWI2 qui consiste d'abord à recentrer le compteur programme (2 fois DEC 11,S) puis à remettre l'instruction d'origine sauvegardée en Memo 07DF,EO. Un branchement à RSWI permet de visualiser et éventuellement de changer le contenu des registres du µp. Après retour à R POINT ou après RESET, l'appui sur la tou-

Initialisation de la fonction SWI2

```
E27C 6A   6B          A RSWI2  DEC   11,S
E27E 6A   6B          A        DEC   11,S
E280 FC   07DF        A        LDD   SASWI2
E283 ED   F8 0A       A        STD   [10,S]
E286 20   1F   E2A7            BRA   RSWI
```

Place un point d'arrêt, sauve l'instruction

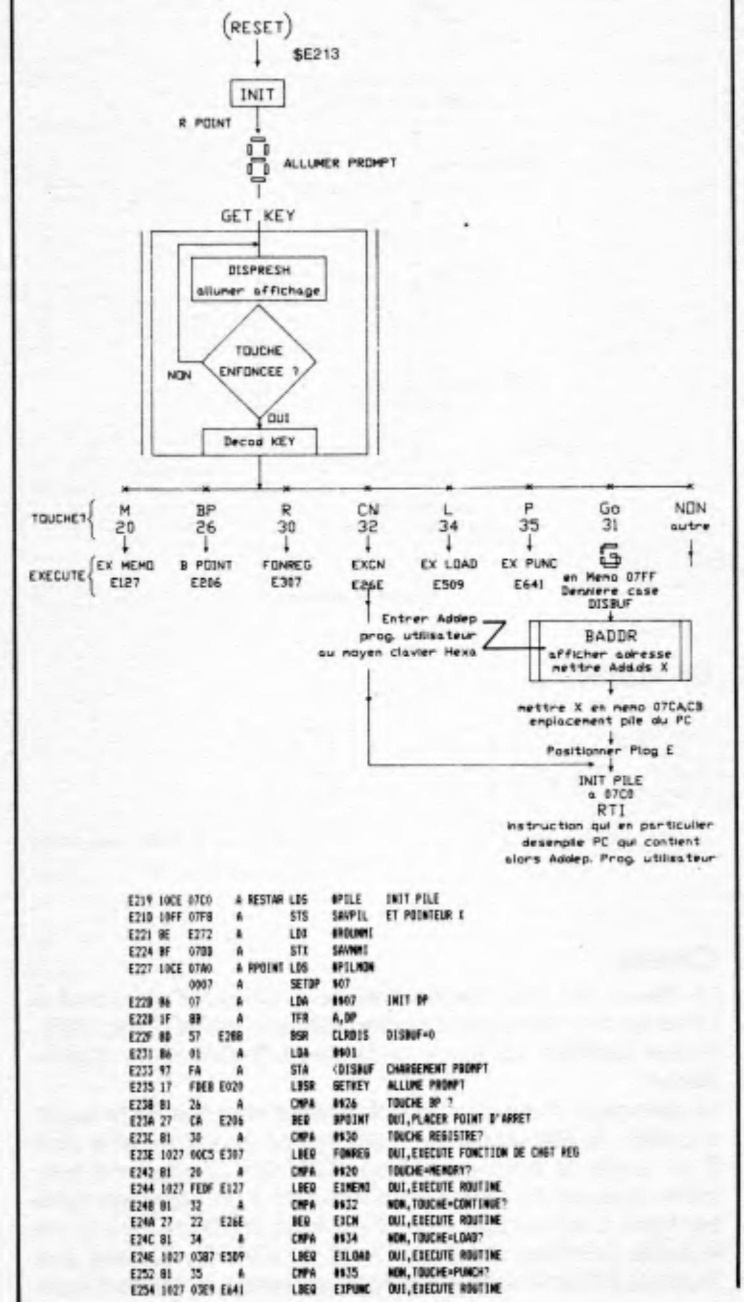
```
E206 CC   756B        A BPOINT LDD   #$756B
E209 DD   FE          A        STD   <  DISBUF+4
E20B 17   FEA7 E0B5            LBSR  BADDR
E20E EC   84          A        LDD   ,X
E210 DD   DF          A        STD   <  SASWI2
E212 CC   103F        A        LDD   #$103F
E215 ED   84          A        STD   ,X
E217 20   0E   E227            BRA   RPOINT
```

che CN permet de relancer le programme utilisateur à partir de l'endroit où il s'était arrêté.

Nous conseillons aux lecteurs de revenir à l'étude de ces deux fonctions d'interruption après s'être familiarisé avec les sous-programmes GETKEY, DISPRESH et BADDR.

Programme «Reset»

Initialisation, décodage des touches M, R, CN, L, P, GO et lancement du programme utilisateur



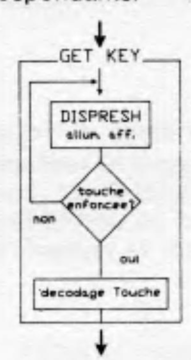

```
E219 10CE 07C0   A RESTAR LDS   #PILE    INIT PILE
E21D 10FF 07F8   A        STS   SAVPIL   ET POINTEUR X
E221 8E   E272   A        LDX   #ROUNMI
E224 BF   07DD   A        STX   SAVNMI
E227 10CE 07A0   A RPOINT LDS   #PILMON
          0007   A        SETDP $07
E22B 86   07     A        LDA   #$07     INIT DP
E22D 1F   8B     A        TFR   A,DP
E22F 8D   57   E288       BSR   CLRDIS   DISBUF=0
E231 86   01     A        LDA   #$01
E233 97   FA     A        STA   <DISBUF  CHARGEMENT PROMPT
E235 17   FDE8 E020       LBSR  GETKEY   ALLUME PROMPT
E238 81   26     A        CMPA  #$26     TOUCHE BP ?
E23A 27   CA   E206       BEQ   BPOINT   OUI,PLACER POINT D'ARRET
E23C 81   30     A        CMPA  #$30     TOUCHE REGISTRE?
E23E 1027 00C5 E307       LBEQ  FONREG   OUI,EXECUTE FONCTION DE CHGT REG
E242 81   20     A        CMPA  #$20     TOUCHE=MEMORY?
E244 1027 FEDF E127       LBEQ  EXMEMO   OUI,EXECUTE ROUTINE
E248 81   32     A        CMPA  #$32     NON,TOUCHE=CONTINUE?
E24A 27   22   E26E       BEQ   EXCN     OUI,EXECUTE ROUTINE
E24C 81   34     A        CMPA  #$34     NON,TOUCHE=LOAD?
E24E 1027 0387 E5D9       LBEQ  EXLOAD   OUI,EXECUTE ROUTINE
E252 81   35     A        CMPA  #$35     NON,TOUCHE=PUNCH?
E254 1027 03E9 E641       LBEQ  EXPUNC   OUI,EXECUTE ROUTINE
E258 81   31     A        CMPA  #$31     NON,TOUCHE=GO?
E25A 26   CB   E227       BNE   RPOINT   NON,RETOUR SCRUTATION
E25C 86   7C     A        LDA   #$7C     OUI,CHARGE G DS DERNIER DIGIT
E25E 97   FF     A        STA   <DISBUF+5
E260 17   FE52 E0B5       LBSR  BADDR    FABRIQUE ADRESSE DEPART
E263 109E F8     A        LDY   <SAVPIL  DU PROGRAMME
E266 AF   2A     A        STX   10,Y     ADRESSE PROG DANS PC
E268 86   80     A        LDA   #$80
E26A AA   A4     A        ORA   ,Y       POSITIONNER FLAG E=1
E26C A7   A4     A        STA   ,Y       POUR PRENDRE EN COMPTE
E26E 10DE F8     A EXCN   LDS   <SAVPIL  TOUS LES REGISTRES
E271 3B                   RTI            DEPART PROG UTILISATEUR
```

Affichage & Clavier

Ces deux fonctions utilisent les mêmes lignes du PIA. En outre, les deux sous-programmes :

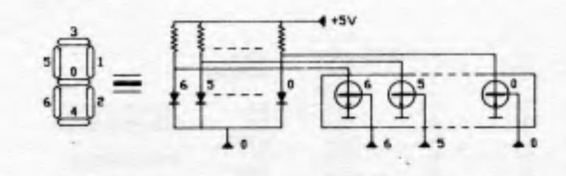
- Affichage (DISPRESH)
- Clavier (GET KEY)

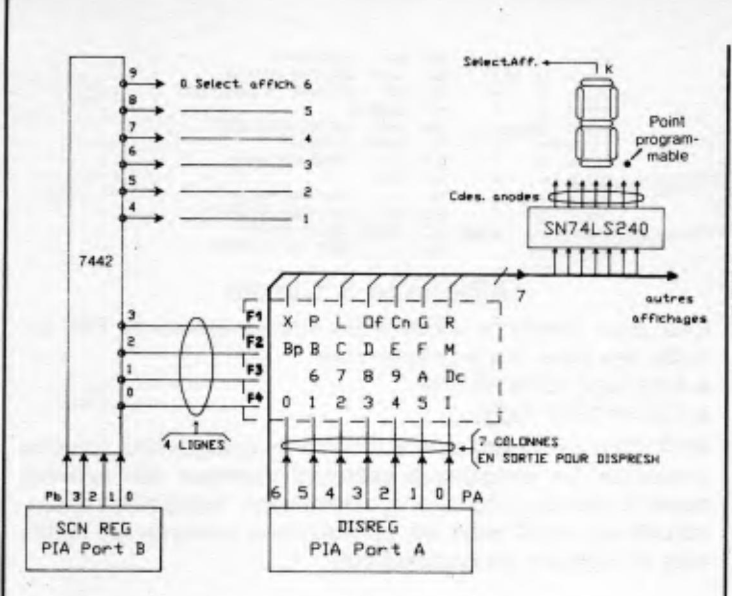
sont imbriqués ainsi que le montre l'organigramme général ci-contre. Le programme principal moniteur fait souvent appel à ces deux fonctions essentielles. Aussi nous analyserons en détail dans les paragraphes suivants les matériels et logiciels correspondants.



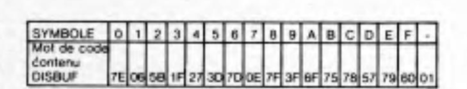
Affichage

Il comprend 6 afficheurs à cathode commune. Pour des raisons évidentes de simplicité, les données - ou de façon plus générale - les symboles à afficher ne sont pas mémorisés mais multiplexés. Un rafraîchissement (par programme) est dès lors nécessaire. Les six symboles à afficher sont (après codage 7 segments) rangés dans **6 positions mémoires** consécutives d'adresses **07FA à 07FF** (appelées **DISBUF**). Après lecture de DISBUF et aiguillage via le PIA port A, chaque bit de symbole commande l'exctinction ou l'allumage d'un segment lumineux par dérivation ou non du courant d'alimentation des LEDS dans le transistor correspondant du 74LS240 (voir ci-dessous). Ceci n'est possible que si l'afficheur est sélectionné. La sélection s'effectue à partir du PIA port B (SCN REG) décodé par le 7442 qui porte la cathode de l'afficheur à zéro.

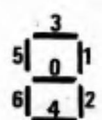




Sachant qu'un «0» à l'entrée du 74240 allume 1 segment lumineux (transistor bloqué) et sachant que le contenu des positions mémoires DISBUF est complémenté par programme (Adresse E0A4 de DISPRESH) avant d'être aiguillés par le PIA, il est facile de vérifier la correspondance suivante :



SYMBOLE	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	.
Mot de code contenu DISBUF	7E	06	5B	1F	27	3D	7D	0E	7F	3F	6F	75	78	57	79	60	01

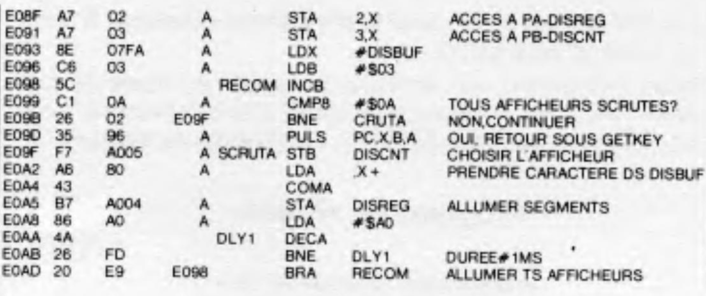


C'est la table DIGTBL. Les différents codes 7 segments (7E, 06... 69) sont rangés en mémoire à partir de l'adresse E010.

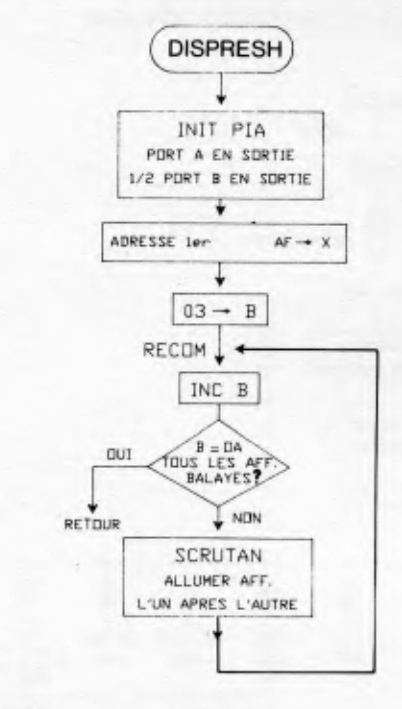
Quant à la partie programme DISPRESH, elle ne présente pas de difficulté majeure. Le PIA port A (DISREG) est programmé en sortie afin d'aiguiller les différents mots de commande en provenance de DISBUF. Un demi-port (port B) suffit pour sélectionner les 6 afficheurs par l'intermédiaire du décodeur 7442 (3 bits suffiraient mais il faut aussi adresser 4 lignes de clavier comme nous le verrons dans le § suivant). L'adressage indexé facilite le balayage séquentiel des afficheurs. L'afficheur sélectionné illumine le symbole voulu durant 1 ms.

Allumage des afficheurs

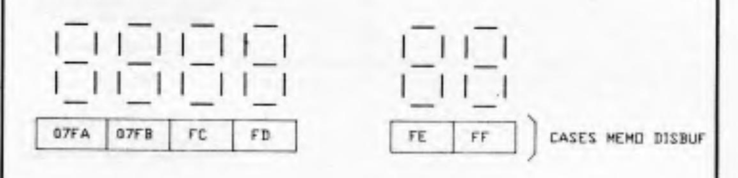
```
E07B 34   16      A DISPRE PSHS  X,B,A
E07D BE   A004    A        LDX   #DISREG
E080 4F                    CLRA
E081 A7   02      A        STA   2,X       ACCES A DDRA
E083 A7   03      A        STA   3,X       ACCES A DDRB
E085 86   7F      A        LDA   #$7F
E087 A7   84      A        STA   ,X        PA EN SORTIE
E089 86   0F      A        LDA   #$0F
E08B A7   01      A        STA   1,X       PBO-3 EN SORTIE
E08D 86   04      A        LDA   #$04
```



```
E08F A7   02      A        STA   2,X       ACCES A PA-DISREG
E091 A7   03      A        STA   3,X       ACCES A PB-DISCNT
E093 8E   07FA    A        LDX   #DISBUF
E096 C6   03      A        LDB   #$03
E098 5C             RECOM  INCB
E099 C1   0A      A        CMP8  #$0A      TOUS AFFICHEURS SCRUTES?
E09B 26   02   E09F        BNE   CRUTA     NON,CONTINUER
E09D 35   96      A        PULS  PC,X,B,A  OUI, RETOUR SOUS GETKEY
E09F F7   A005    A SCRUTA STB   DISCNT    CHOISIR L'AFFICHEUR
E0A2 A6   80      A        LDA   ,X+       PRENDRE CARACTERE DS DISBUF
E0A4 43                    COMA
E0A5 B7   A004    A        STA   DISREG    ALLUMER SEGMENTS
E0A8 86   A0      A        LDA   #$A0
E0AA 4A             DLY1   DECA
E0AB 26   FD               BNE   DLY1      DUREE#1MS
E0AD 20   E9   E098        BRA   RECOM     ALLUMER TS AFFICHEURS
```



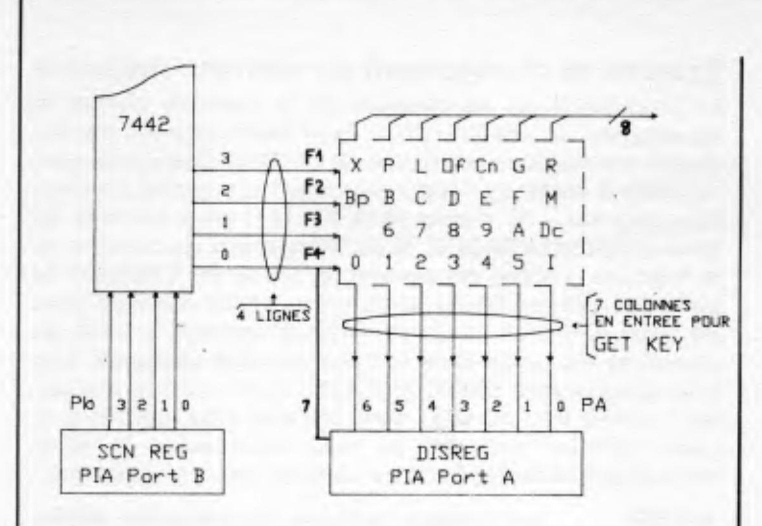
On rappelle :



Clavier

Le clavier est organisé en matrice 4 lignes × 7 colonnes, à l'intersection desquelles se trouvent les contacts des différentes touches. La figure ci-dessous illustre cette organisation.

La détection d'une touche enfoncée s'effectue de la façon suivante : le PIA port A étant positionné en entrée et le port B en sortie, le sous-programme GETKEY réalise une première analyse du clavier en procédant à un balayage ligne par ligne. L'adressage d'un n° de ligne par SCN REG porte la sortie correspondante du 7442 à l'état «0». Si l'une des touches est enfoncée sur la ligne adressée, le contact ligne

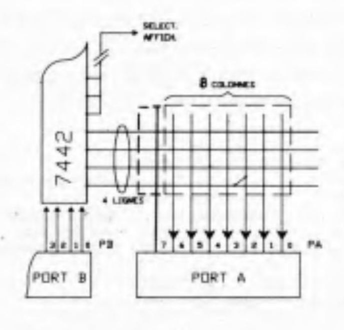


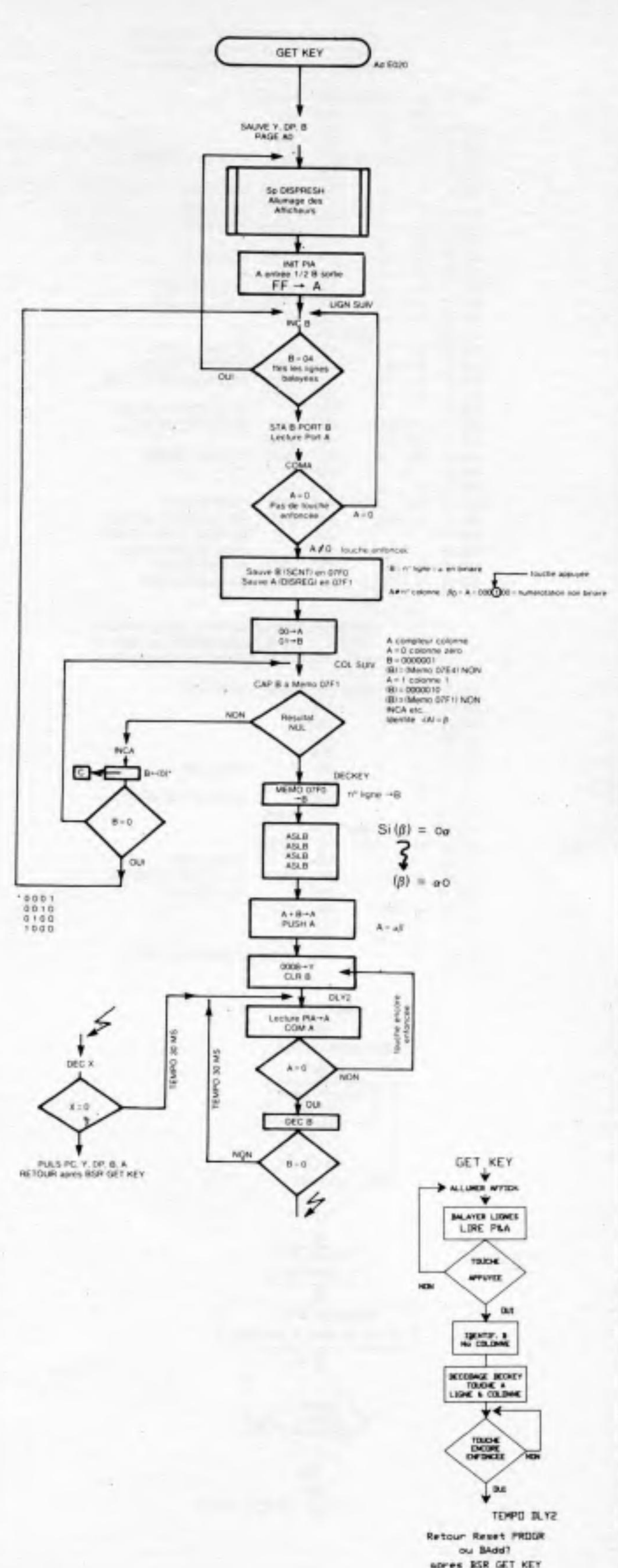
colonne impose cet état «0» sur l'une des colonnes, donc sur l'une des entrées du PIA DISREG. Une simple lecture du PIA port A nous renseigne sur l'état du clavier. Compte tenu de la complémentation de la lecture du PIA (Ad EO44 de GETKEY) une lecture = FF traduit l'appui sur l'une des touches. Il faut alors procéder à l'identification de la touche actionnée. Si l'état du port B donne la position de la touche en ligne, la position colonne est obtenue en repérant la position du «1» (ne pas oublier la complémentation) des données du port A : c'est le rôle de la partie DECKEY qui se termine par la mise dans l'accumulateur A du code touche. L'organigramme très détaillé présenté ci-après permet de comprendre au mieux le sous-programme GETKEY. Si aucune touche n'est actionnée, le programme repart en rafraîchissant les afficheurs pour revenir à la détection d'une touche et ainsi de suite.

Le code $\alpha\beta$ - code touche - correspond à la α^e ligne et à la β^e colonne. Par exemple la touche [I] est codée 00. Les divers codes-touches sont rangés en mémoire 2716 à partir de l'adresse E000 dans l'ordre suivant :

n° touche	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Code touche	06	05	04	03	02	01	15	14	13	12	11	25	24	23	22	21

C'est la table KEYTBL. Cette table ne comprend que les codes des touches HEXA. Le lecteur établira facilement le code des touches de fonction (Inc Dec Mem...) (c'est-à-dire 00, 10, 20...).



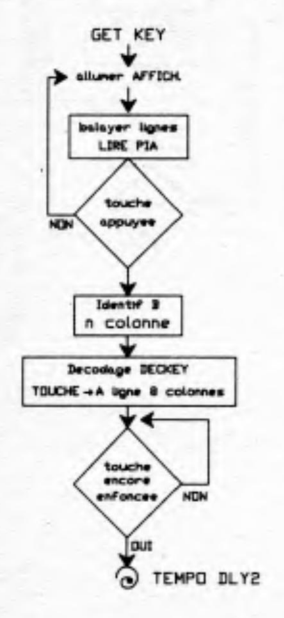


```
**************** GETKEY ROUTINE ***************
*SCRUTE LES LIGNES ET LES COLONNES DU CLAVIER*
************* ALLUME LES AFFICHEURS ************

E020 34   2C       A GETKEY PSHS   Y,DP,B
          00A0     A        SETDP  $A0
E022 86   A0       A        LDA    #$A0
E024 1F   8B       A        TFR    A,DP
E026 8D   53   E07B FINCLA  BSR    DISPRE   ALLUMER LES AFFICHEURS
E028 4F                     CLRA
E029 97   06       A        STA    <SCNREG  ACCES A DDRA
E02B 97   07       A        STA    <SCNCNT  ACCES A DDRB
E02D 97   04       A        STA    <DISREG  PA EN ENTREE
E02F 86   0F       A        LDA    #$0F
E031 97   05       A        STA    <DISCNT  PB EN SORTIE
E033 86   04       A        LDA    #$04
E035 97   06       A        STA    <SCNREG  ACCES A ORA-DISREG
E037 97   07       A        STA    <SCNCNT  ACCES A ORB-DISCNT
E039 C6   FF       A        LDB    #$FF
E03B 5C                LIGSUI INCB
E03C C1   04       A        CMPB   #$04     FIN SCRUT. CLAVIER?
E03E 27   E6   E026         BEQ    FINCLA   OUI,SCRUTER AFFICHEURS
E040 D7   05       A        STB    <DISCNT  NON,SCRUTER LIGNE PAR LIGNE
E042 96   04       A        LDA    <DISREG
E044 43                     COMA            PAS DE TOUCHES ENFONCEES?
E045 27   F4   E03B         BEQ    LIGSUI   OUI,SCRUTER LIGNE SUIV.
E047 F7   07F0     A        STB    SAVCNT
E04A B7   07F1     A        STA    SAVREG   NON,TOUCHE ENFONCEE
E04D 4F                     CLRA
E04E C6   01       A        LDB    #$01
E050 F1   07F1     A COLSUI CMPB   SAVREG   TOUCHE DETECTEE?
E053 27   06   E05B         BEQ    DECKEY   OUI, RECONNAITRE LA TOUCHE
E055 4C                     INCA            NON,PASSER COLONNE SUIVANTE
E056 58                     ASLB            TOUTES COLONNES TESTEES?
E057 27   E2   E03B         BEQ    LIGSUI   OUI,LIGNE SUIVANTE
E059 20   F5   E050         BRA    COLSUI   NON,COLONNE SUIVANTE
```

```
************ RECONNAISSANCE DE LA TOUCHE ***********
*FABRIQUE LE CODE DE RECONNAISSANCE DE LA TOUCHE SI*
** ELLE EST APPUYEE ; SUPPRIME LES REBONDISSEMENTS**

E05B F6   07F0     A DECKEY LDB    SAVCNT   NUMERO LIGNE
E05E 58                     ASLB
E05F 58                     ASLB
E060 58                     ASLB
E061 58                     ASLB
E062 34   04       A        PSHS   B        REPERE LIGNE
E064 AB   E0       A        ADDA   ,S+
E066 34   02       A        PSHS   A        SAUVEGARDE CODE TOUCHE
E068 108E 0008     A        LDY    #$0008
E06C 5F                NOREB CLRB
E06D 96   04       A DLY2   LDA    <DISREG
E06F 43                     COMA            REBONDISSEMENT?
E070 26   FA   E06C         BNE    NOREB    OUI, ATTENDRE DISPARITION
E072 5A                     DECB            NON, TEMPO=30MS
E073 26   F8   E06D         BNE    DLY2
E075 31   3F       A        LEAY   -1,Y
E077 26   F4   E06D         BNE    DLY2
E079 35   AE       A        PULS   PC,Y,DP,B,A RETOUR RESET ROUTINE
```



Examen et changement du contenu mémoire

Le programme de visualisation de la mémoire permet la visualisation du contenu de la case Memo pointée par les quatre premières cases Memo de DISBUF. Ces cases sont remplies à partir de l'appui successif sur quatre touches hexadécimales du clavier (H H' H'' H''') sous contrôle du sous-programme BADDR. Si un changement du contenu de la mémoire s'avère nécessaire, la partie de EXMEMO (à partir de l'adresse E155) prend les nouvelles données dans les deux dernières cases de DISBUF (entrées à partir du clavier) et les range dans la case mémoire désignée. Les sous-programmes EXINC et EXDEC sont appelés lorsque les touches INC ou DEC sont utilisées pour passer à la case mémoire suivante ou pour revisualiser la case mémoire précédente. Dans ce chapitre, nous analyserons :

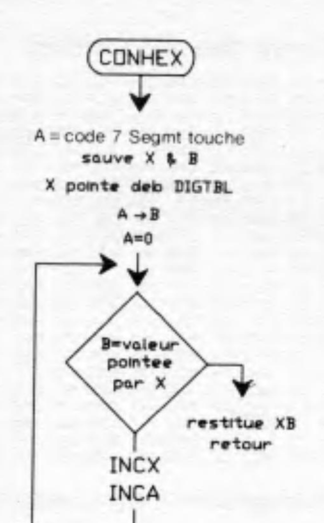
BADDR	qui fabrique l'adresse hexadécimale entrée à partir du clavier, rafraîchit l'affichage et met cette adresse dans X
EXMEMO	qui permet la visualisation (et le changement si besoin) du contenu de la case mémoire pointée par X.
EXINC EXDEC	qui incrémente ou décrémente l'adresse visualisée. Une lecture de la case hexa a lieu avant de ranger la donnée (on a pu essayer d'écrire en EPROM).

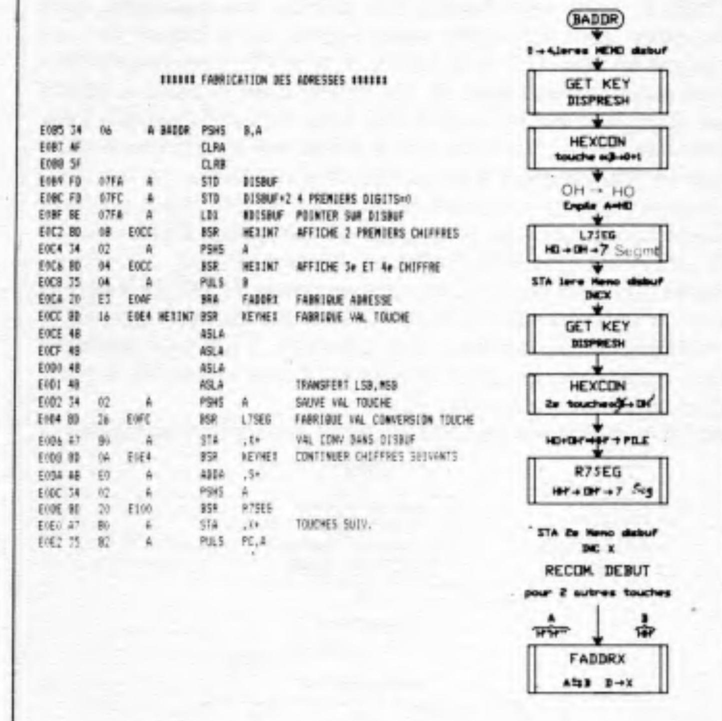
Compte tenu du nombre de sous-programmes mis en jeu, nous proposons le petit aide-mémoire suivant :

HEX CON	Convertit 1 code touche ($\alpha\beta$) en son équivalent hexa et le remet dans A. Si la touche n'est pas hexadecimale, le programme se branche à R POINT.
L75EG	Dans le cas de deux chiffres hexa (dans A), convertit le chiffre de gauche en code 7 segments (compatible afficheurs) et le place dans A.
CON HEX	(7 TOHEX) convertit le code 7 segments (dans A) en son équivalent hexadécimal et le place dans A.
KEY HEX	combine GETKEY & HEXCON
HEXIN 7	Utilise KEYHEX pour accepter l'entrée de deux touches et combine les deux valeurs hexa (OH & OH') en 1 seule (H H') dans A.

Nous précisons encore que GETKEY détecte 1 touche hexa sous la forme $\alpha\beta$ (α ligne, β colonne). Cette touche peut être convertie en son équivalent hexa OH par HEXCON. Elle peut être transcodée hexa en code 7 segments compatible afficheur par R7SEG. Elle peut aussi être relue en hexa à partir de disbuf par CONHEX.

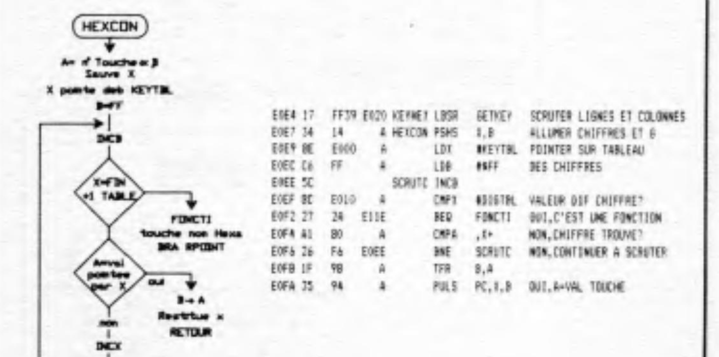
```
E112 34   14       A CONHEX PSHS   X,B
E114 8E   E010     A        LDX    #DIGTBL
E117 1F   89       A        TFR    A,B      B=DISBUF+4 OU DISBUF+5
E119 4F                     CLRA
E11A E1   80       A NONFIN CMPB   ,X+      CHERCHER VAL DONNEE
E11C 27   07   E125         BEQ    DONEA    A=VAL HEXA DONNEE
E11E 1027 0105 E227 FONCTI LBEQ   RPOINT   OUI,FONCTION,RETOURNER SCRUTER
E122 4C                     INCA
E123 20   F5   E11A         BRA    NONFIN
E125 35   94       A DONEA  PULS   PC,X,B
```





FABRICATION DES ADRESSES

```
E0B5 34    06       A BADDR  PSHS   B,A
E0B7 4F                      CLRA
E0B8 5F                      CLRB
E0B9 FD    07FA     A        STD    DISBUF
E0BC FD    07FC     A        STD    DISBUF+2 4 PREMIERS DIGITS=0
E0BF 8E    07FA     A        LDX    #DISBUF  POINTER SUR DISBUF
E0C2 8D    08    E0CC        BSR    HEXIN7   AFFICHE 2 PREMIERS CHIFFRES
E0C4 34    02       A        PSHS   A
E0C6 8D    04    E0CC        BSR    HEXIN7   AFFICHE 3e ET 4e CHIFFRE
E0C8 35    04       A        PULS   B
E0CA 20    E3    E0AF        BRA    FADDRX   FABRIQUE ADRESSE
E0CC 8D    16    E0E4 HEXIN7 BSR    KEYHEX   FABRIQUE VAL TOUCHE
E0CE 48                      ASLA
E0CF 48                      ASLA
E0D0 48                      ASLA
E0D1 48                      ASLA            TRANSFERT LSB,MSB
E0D2 34    02       A        PSHS   A        SAUVE VAL TOUCHE
E0D4 8D    26    E0FC        BSR    L7SEG    FABRIQUE VAL CONVERSION TOUCHE
E0D6 A7    80       A        STA    ,X+      VAL CONV DANS DISBUF
E0D8 8D    0A    E0E4        BSR    KEYHEX   CONTINUER CHIFFRES SUIVANTS
E0DA AB    E0       A        ADDA   ,S+
E0DC 34    02       A        PSHS   A
E0DE 8D    20    E100        BSR    R7SEG
E0E0 A7    80       A        STA    ,X+      TOUCHES SUIV.
E0E2 35    82       A        PULS   PC,A
```



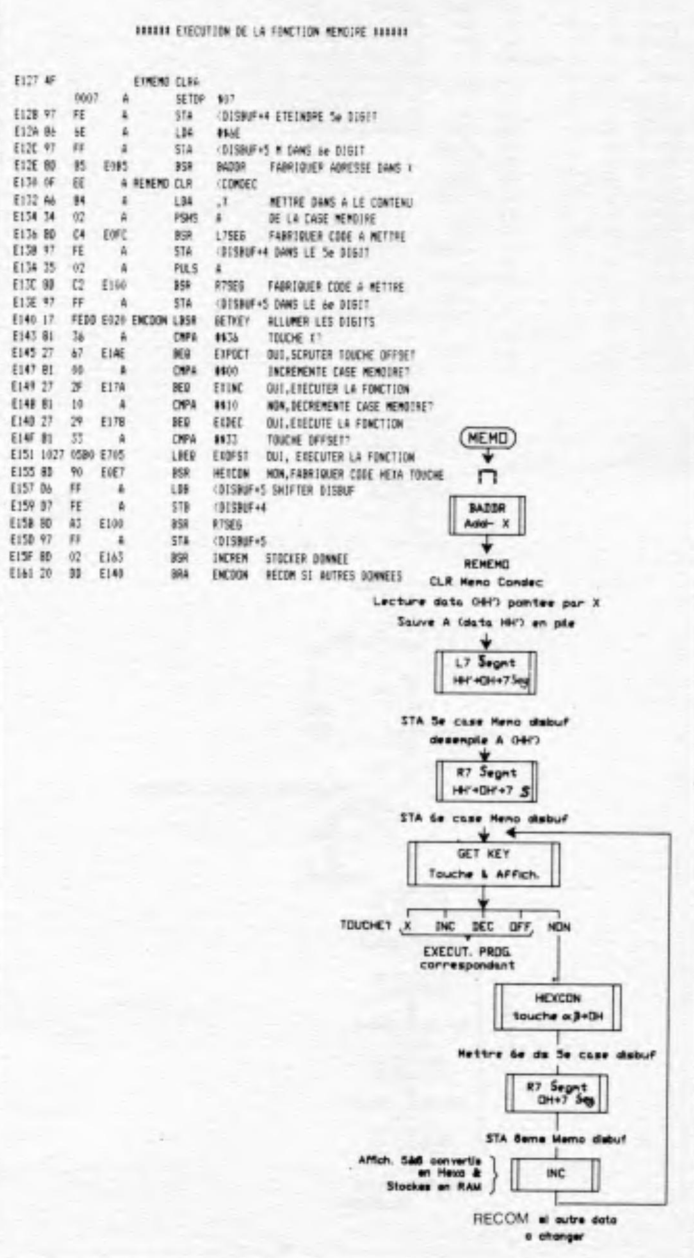

```
E0E4 17    FF39  E020 KEYHEX LBSR   GETKEY   SCRUTER LIGNES ET COLONNES
E0E7 34    14       A HEXCON PSHS   X,B      ALLUMER CHIFFRES ET 6
E0E9 8E    E000     A        LDX    #KEYTBL  POINTER SUR TABLEAU
E0EC C6    FF       A        LDB    #$FF     DES CHIFFRES
E0EE 5C               SCRUTC INCB
E0EF 8C    E010     A        CMPX   #DIGTBL  VALEUR DIF CHIFFRE?
E0F2 27    2A    E11E        BEQ    FONCTI   OUI,C'EST UNE FONCTION
E0F4 A1    80       A        CMPA   ,X+      NON,CHIFFRE TROUVE?
E0F6 26    F6    E0EE        BNE    SCRUTC   NON,CONTINUER A SCRUTER
E0F8 1F    98       A        TFR    B,A
E0FA 35    94       A        PULS   PC,X,B   OUI,A=VAL TOUCHE
```

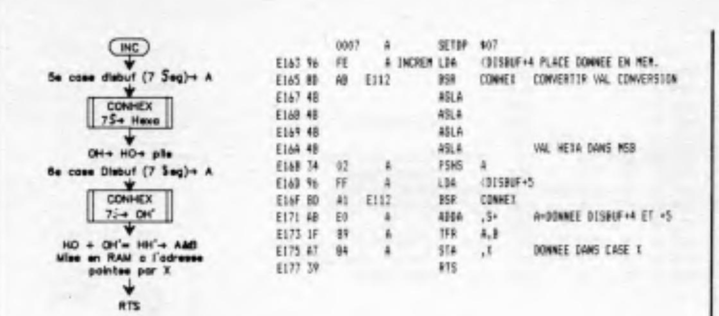
```
E0FC 47               L7SEG  ASRA
E0FD 47                      ASRA
E0FE 47                      ASRA
E0FF 47                      ASRA            VAL TOUCHE DS A , LSB
E100 34    10       A R7SEG  PSHS   X
E102 8E    E010     A        LDX    #DIGTBL  POINTER SUR TABLEAU
E105 84    0F       A        ANDA   #$0F     PREMIERE TOUCHE?
E107 27    05    E10E NOVALH BEQ    VALHEX   OUI,VAL HEX DS A
E109 30    01       A        LEAX   1,X      NON,POINTER SUR VAL SUIVANTE
E10B 4A                      DECA
E10C 20    F9    E107        BRA    NOVALH   RECOMMENCER SI VAL NON TROUVEE
E10E A6    84       A VALHEX LDA    ,X       CONVERSION HEXA-7SEGMENTS
E110 35    90       A        PULS   PC,X     DANS A TROUVEE
```

```
E0AF 1E    89       A FADDRX EXG    A,B
E0B1 1F    01       A        TFR    D,X      B+X=ADRESSE PROGRAMME
E0B3 35    86       A        PULS   PC,B,A
```

EXECUTION DE LA FONCTION MEMOIRE

```
E127 4F               EXMEMO CLRA
           0007     A        SETDP  $07
E128 97    FE       A        STA    <DISBUF+4 ETEINDRE 5e DIGIT
E12A 86    6E       A        LDA    #$6E
E12C 97    FF       A        STA    <DISBUF+5 M DANS 6e DIGIT
E12E 8D    85    E0B5        BSR    BADDR    FABRIQUER ADRESSE DANS X
E130 0F    EE       A REMEMO CLR    <COMDEC
E132 A6    84       A        LDA    ,X       METTRE DANS A LE CONTENU
E134 34    02       A        PSHS   A        DE LA CASE MEMOIRE
E136 8D    C4    E0FC        BSR    L7SEG    FABRIQUER CODE A METTRE
E138 97    FE       A        STA    <DISBUF+4 DANS LE 5e DIGIT
E13A 35    02       A        PULS   A
E13C 8D    C2    E100        BSR    R7SEG    FABRIQUER CODE A METTRE
E13E 97    FF       A        STA    <DISBUF+5 DANS LE 6e DIGIT
E140 17    FEDD  E020 ENCDON LBSR   GETKEY   ALLUMER LES DIGITS
E143 81    36       A        CMPA   #$36     TOUCHE X?
E145 27    67    E1AE        BEQ    EXPOCT   OUI,SCRUTER TOUCHE OFFSET
E147 81    00       A        CMPA   #$00     INCREMENTE CASE MEMOIRE?
E149 27    2F    E17A        BEQ    EXINC    OUI,EXECUTER LA FONCTION
E14B 81    10       A        CMPA   #$10     NON,DECREMENTE CASE MEMOIRE?
E14D 27    29    E178        BEQ    EXDEC    OUI,EXECUTE LA FONCTION
E14F 81    33       A        CMPA   #$33     TOUCHE OFFSET?
E151 1027  05B0  E705        LBEQ   EXOFST   OUI, EXECUTER LA FONCTION
E155 8D    90    E0E7        BSR    HEXCON   NON,FABRIQUER CODE HEXA TOUCHE
E157 D6    FF       A        LDB    <DISBUF+5 SHIFTER DISBUF
E159 D7    FE       A        STB    <DISBUF+4
E15B 8D    A3    E100        BSR    R7SEG
E15D 97    FF       A        STA    <DISBUF+5
E15F 8D    02    E163        BSR    INCREM   STOCKER DONNEE
E161 20    DD    E140        BRA    ENCDON   RECOM SI AUTRES DONNEES
```






```
            0007     A          SETDP  $07
E163 96     FE       A INCREM   LDA    <DISBUF+4 PLACE DONNEE EN MEM.
E165 8D     AB  E112            BSR    CONHEX    CONVERTIR VAL CONVERSION
E167 48                         ASLA
E168 48                         ASLA
E169 48                         ASLA
E16A 48                         ASLA             VAL HEXA DANS MSB
E16B 34     02       A          PSHS   A
E16D 96     FF       A          LDA    <DISBUF+5
E16F 8D     A1  E112            BSR    CONHEX
E171 AB     E0       A          ADDA   ,S+       A=DONNEE DISBUF+4 ET +5
E173 1F     89       A          TFR    A,B
E175 A7     84       A          STA    ,X        DONNEE DANS CASE X
E177 39                         RTS
```

```
        ######## EXECUTION DE LA FONCTION INCREMENTATION ########
        ############# ET DE LA FONCTION DECREMENTATION ############

E178 0A   EE         A EXDEC  DEC   <COMDEC
E17A 8D   E7    E163 EXINC    BSR   INCREM
E17C 96   EE         A        LDA   <COMDEC  INCREMENTE OU DECREMENTE?
E17E 26   28    E1A8          BNE   DECRE    COMDEC<>0,DECREMENTE
E180 A6   80         A        LDA   ,X+      PRENDRE DONNEE DS X ET X+1
E182 34   04         A PREXIN PSHS  B        SAUVE AVANT STOCKAGE
E184 A1   E0         A        CMPA  ,S+      MEMOIRE ABSENTE OU MEMOIRE MORTE?
E186 1026 009D  E227          LBNE  RPOINT   OUI,ALLUMER PROMPT ET SCRUTER
E18A 1F   10         A        TFR   X,D      NON,AFFICHE CASE MEMOIRE
E18C 17   FF6D  E0FC          LBSR  L7SEG    SUIVANTE OU PRECEDENTE
E18F 97   FA         A        STA   <DISBUF
E191 1F   10         A        TFR   X,D
E193 17   FF6A  E100          LBSR  R7SEG
E196 97   FB         A        STA   <DISBUF+1
E198 1F   98         A        TFR   B,A
E19A 17   FF5F  E0FC          LBSR  L7SEG
E19D 97   FC         A        STA   <DISBUF+2
E19F 1F   98         A        TFR   B,A
E1A1 17   FF5C  E100          LBSR  R7SEG
E1A4 97   FD         A        STA   <DISBUF+3
E1A6 20   88    E130          BRA   REMEMO   RECOMMENCER EXECUTION MEMOIRE
E1A8 A6   84         A DECRE  LDA   ,X
E1AA 30   1F         A        LEAX  -1,X
E1AC 20   D4    E182          BRA   PREXIN   DECREMENTATION EXECUTEE
```

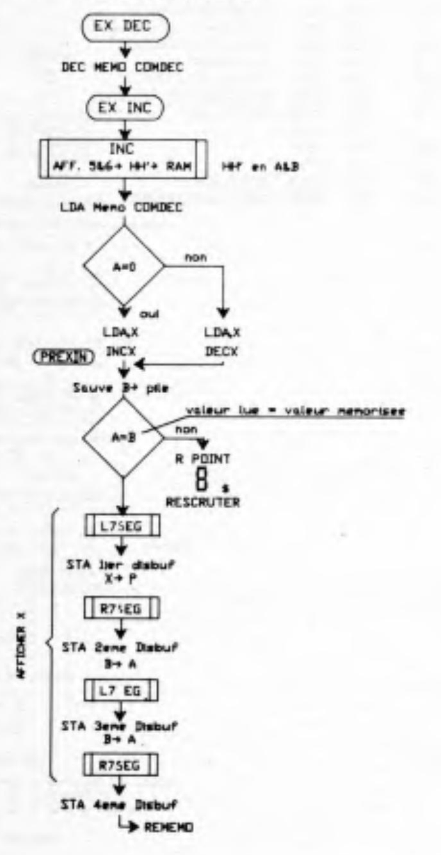


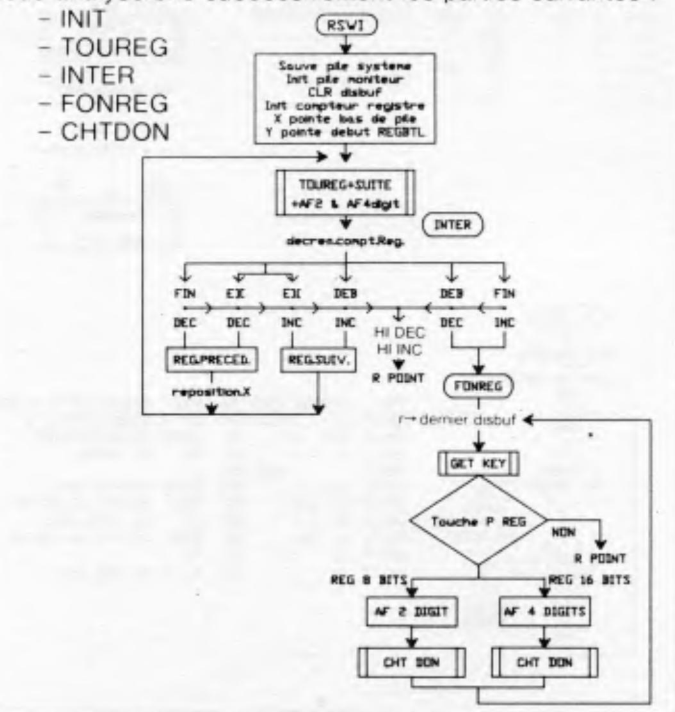
Table des étiquettes

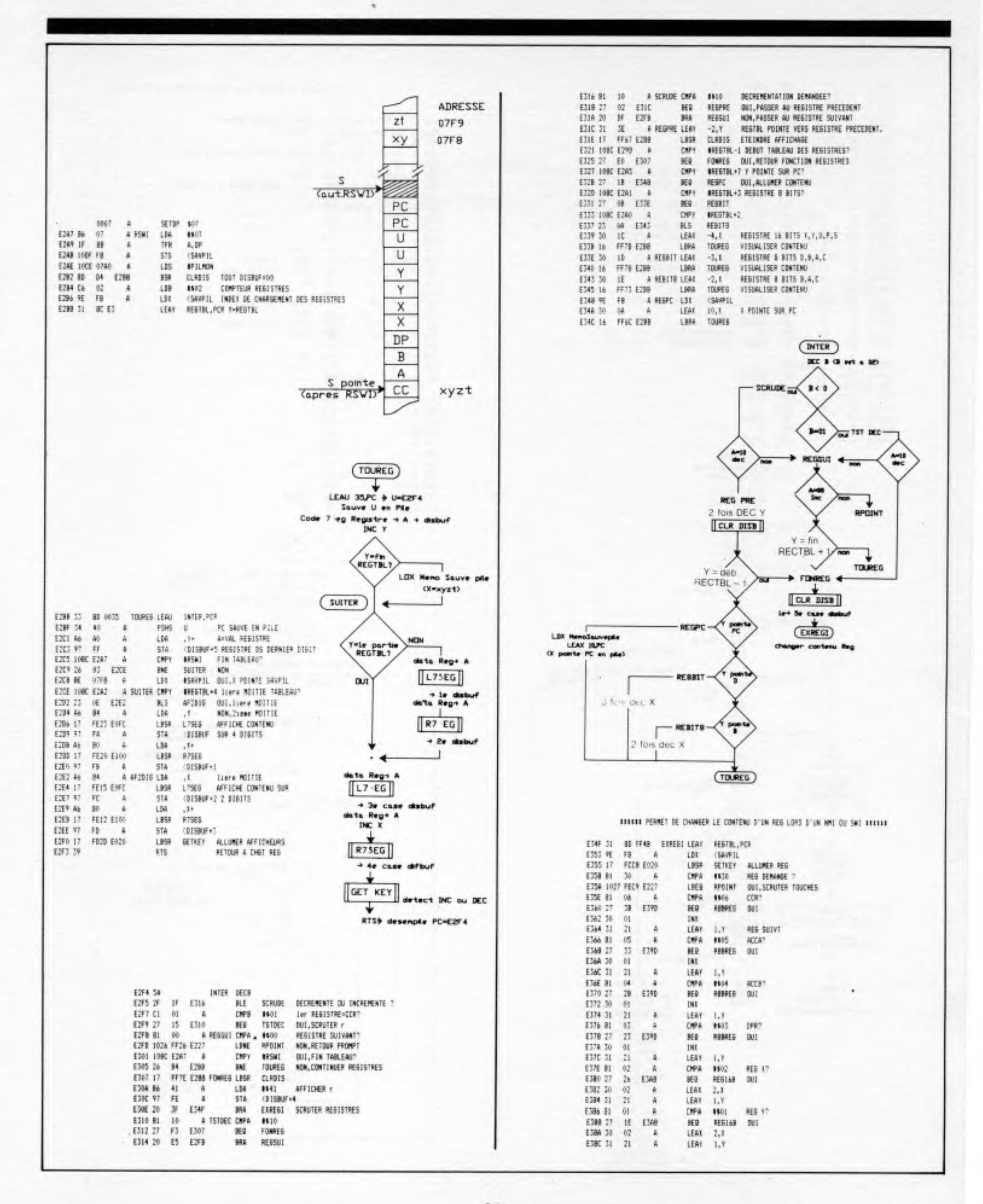
ADDARR 07EC	ADDDEP 07EA	AF2DIG E2E2	AFCLIG E789	AFEROR E56A	AFFIAD E7C8	AFFIAR E763	AFICHA E6[illegible]	AFIVAL E3DD
AFSIGN E58D	BADDR E085	BCOURT E78A	BITCA0 E5D3	BITCA1 E5C5	BITLSB E5C3	BOUCL1 E6F1	BOUCL2 E6F9	BOUCL3 E6E2
BOUFIN E63C	BPOINT E206	BRANNE E72D	BRAMPO E726	CAL16 E4DB	CALCON E404	CALCOR E4EC	CALCUL E5[illegible]	CALOFS E757
CALPOT E1E1	CARRY1 E5B3	CARSUI E605	CHAN16 E3C8	CHAN68 E3B3	CHAR16 E49F	CHARGE E738	CHPOST E440	CHTCHI E44B
CHTDON E4AE	CLIGNE E7AF	CLIGNO E781	CLRDIS E2BB	COOFIN E675	COLSUI E950	COMDEC 07EE	COMPAD E422	COMPAR E4A3
COMPR2 E516	COMPR3 E522	COMPR4 E534	COMPR5 E550	CONHEX E112	CONROT E5CB	CONTI E201	DEBCHA E6[illegible]	DECENC E575
DECHEX E4F4	DECKEY E058	DECRE E1A8	DECT2S E57B	DEPLAC E4E2	DERADD E62D	DETECS E5F1	DIGTBL E01[illegible]	DISBUF 07F[illegible]
DISCNT A005	DISFIN E637	DISPRE E07B	DISREG A004	DLY1 E0AA	DLY2 E06D	DLY300 E5A5	DLY5MS E6DA	DONEA E125
ENCDON E140	ENCORE E296	ERECRE E428	ERREUR E426	ERROR E79E	EXCN E26E	EXDEC E178	EXINC E17A	EXLBCL* E740
EXLBIL E742	EXLOAD E5D9	EXMEMO E127	EXOFST E705	EXPOCT E1AE	EXPUNC E641	EXREGI E34F	FADDR1 E0[illegible]F	FINCAL E571
FINCHA E6C5	FINCLA E026	FINENR E6A4	FONCTI E11E	FONREG E307	GETKEY E020	HEXCON E0E7	HEXIN7 E0CC	INCREM E163
INTER E2F4	INTER2 E29C	INTER3 E597	INTR05 E434	INTR08 E467	INTR16 E488	INTVAL E48F	KEYHEX E0E4	KEYTBL E000
L7SEG E0FC	LIGSUI E03B	LS8D9A E573	M10000 E548	MOINS E1FB	MSBDON E484	MSBLSB E502	MUL100 E58B	MULFIN E52E
MULPAR E500	MUP100 E591	NEGPOS 07E9	NONFIN E11A	NOREB E06C	NOUBLY E590	NOVALH E107	OFFSET E1C3	OFPOCT E718
PILE 07C0	FILMON 07A0	PLUCOU E784	PLUS E1F4	PLUSMS 07E3	POCPOS E449	POSITF E457	PRELDM 07E7	PRESEH 07E5
PRESEL 07E6	PRESEP 07E1	PREXIN E182	PROMPT E291	R7SEG E100	RBBREG E39D	RCR E484	REBBIT E33E	REBIT8 E343
RECBIT E598	RECMCE E53F	RECOM E098	RECYCL E66B	REG16B E3A8	REGPC E348	REGPRE E33C	REGSUI E2FB	REGTBL E29E
REMEMO E130	RESTAR E219	RETOU1 E790	RETOU2 E794	RETOUR E4E1	RFIRQ 0775	RIRQ 0770	RNMI E7EA	ROUNMI E272
RPOINT E227	RSWI E2A7	RSWI2 E27C	RSWI3 077A	SADDR1 07F2	SADDR2 07F4	SAGMI2 07DF	SAUT E6EB	SAUVER 07E2
SAVB1 07F6	SAVB2 07F7	SAVCNT 07F0	SAVNMI 07DD	SAVFIL 07FB	SAVPOC 07E4	SAVREG 07F1	SAVRES 07EF	SCARRY E59F
SCNCNT A007	SCNREG A006	SCRUDE E316	SCRUTA E09F	SCRUTC E0EE	SECPAR E540	STADON E482	STOCKA E781	STODON E47A
SUI E483	SUICHA E61F	SUITEB E750	SUITEP E4C8	SUITER E2CE	SUIVT E777	TEMPO E7C4	TESTER E557	TESTST E41A
TOTRAN E690	TOUREG E2BE	TRANSM E6D8	TSTDEC E310	TSTPOC E444	TSTSUI E420	VALHEX E10E		

Examen et changement du contenu des registres

Le programme RSWI transfère les registres de la pile dans DISBUF pour que l'opérateur puisse les examiner. Les registres sont visualisés dans l'ordre dans lequel ils sont rangés en pile : CC, A, B, DP, X, Y, U et PC. Une astuce permet aussi de visualiser S. On passe d'un registre à l'autre en appuyant sur la touche INC (reg. suivant) ou DEC (reg. précédent). Le branchement à RSWI est automatique lorsque le déroulement d'un programme utilisateur se heurte à un point d'arrêt (revoir mise en route et init.) ou à un SWI. Dans ce dernier cas, le vecteur d'interruption E27A conduit le µp à exécuter BRA SWI (voir Adresse E27A).
Après examen du contenu des registres, RSWI se poursuit par un branchement à FONREG. Cette fonction permet le changement du contenu des registres. Il est bien entendu que cette fonction peut directement être exécutée à partir d'un appui sur la touche REG.
Nous analyserons successivement les parties suivantes :

- INIT
- TOUREG
- INTER
- FONREG
- CHTDON






```
.            0007     A        SETDP  $07
E2A7 B6   07       A RSWI   LDA    #$07
E2A9 1F   8B       A        TFR    A,DP
E2AB 10DF F8       A        STS    <SAVPIL
E2AE 10CE 07A0     A        LDS    #PILMON
E2B2 8D   D4   E288         BSR    CLRDIS   TOUT DISBUF=00
E2B4 C6   02       A        LDB    #$02     COMPTEUR REGISTRES
E2B6 9E   FB       A        LDX    <SAVPIL  INDEX DE CHARGEMENT DES REGISTRES
E2B8 31   8C E3             LEAY   REGTBL,PCR Y=REGTBL
```

```
E2BB 33   8D 0035   TOUREG LEAU   INTEP,PCR
E2BF 34   40       A        PSHS   U        PC SAUVE EN PILE
E2C1 A6   A0       A        LDA    ,Y+      A=VAL REGISTRE
E2C3 97   FF       A        STA    <DISBUF+5 REGISTRE DS DERNIER DIGIT
E2C5 108C E2A7     A        CMPY   #RSWI    FIN TABLEAU?
E2C9 26   03   E2CE         BNE    SUITER   NON
E2CB 8E   07F8     A        LDX    #SAVPIL  OUI,X POINTE SAVPIL
E2CE 108C E2A2     A SUITER CMPY   #REGTBL+4 1iere MOITIE TABLEAU?
E2D2 23   0E   E2E2         BLS    AF2DIG   OUI,1iere MOITIE
E2D4 A6   84       A        LDA    ,X       NON,2ieme MOITIE
E2D6 17   FE23 E0FC         LBSR   L7SEG    AFFICHE CONTENU
E2D9 97   FA       A        STA    <DISBUF  SUR 4 DIGITS
E2DB A6   80       A        LDA    ,X+
E2DD 17   FE20 E100         LBSR   R7SEG
E2E0 97   FB       A        STA    <DISBUF+1
E2E2 A6   84       A AF2DIG LDA    ,X       1iere MOITIE
E2E4 17   FE15 E0FC         LBSR   L7SEG    AFFICHE CONTENU SUR
E2E7 97   FC       A        STA    <DISBUF+2 2 DIGITS
E2E9 A6   80       A        LDA    ,X+
E2EB 17   FE12 E100         LBSR   R7SEG
E2EE 97   FD       A        STA    <DISBUF+3
E2F0 17   FD2D E020         LBSR   GETKEY   ALLUMER AFFICHEURS
E2F3 39                     RTS             RETOUR A CHGT REG
```

```
E2F4 5A                INTER  DECB
E2F5 2F   1F   E316           BLE    SCRUDE   DECREMENTE OU INCREMENTE ?
E2F7 C1   01       A          CMPB   #$01     1er REGISTRE=CCR?
E2F9 27   15   E310           BEQ    TSTDEC   OUI,SCRUTER r
E2FB 81   00       A REGSUI   CMPA   #$00     REGISTRE SUIVANT?
E2FD 1026 FF26 E227           LBNE   RPOINT   NON,RETOUR PROMPT
E301 108C E2A7     A          CMPY   #RSWI    OUI,FIN TABLEAU?
E305 26   B4   E2BB           BNE    TOUREG   NON,CONTINUER REGISTRES
E307 17   FF7E E288 FONREG    LBSR   CLRDIS
E30A 86   41       A          LDA    #$41     AFFICHER r
E30C 97   FE       A          STA    <DISBUF+4
E30E 20   3F   E34F           BRA    EXREGI   SCRUTER REGISTRES
E310 81   10       A TSTDEC   CMPA   #$10
E312 27   F3   E307           BEQ    FONREG
E314 20   E5   E2FB           BRA    REGSUI
```

```
E316 81   10       A SCRUDE CMPA   #$10     DECREMENTATION DEMANDEE?
E318 27   02   E31C         BEQ    REGPRE   OUI,PASSER AU REGISTRE PRECEDENT
E31A 20   DF   E2FB         BRA    REGSUI   NON,PASSER AU REGISTRE SUIVANT
E31C 31   3E       A REGPRE LEAY   -2,Y     REGTBL POINTE VERS REGISTRE PRECEDENT.
E31E 17   FF67 E288         LBSR   CLRDIS   ETEINDRE AFFICHAGE
E321 108C E29D     A        CMPY   #REGTBL-1 DEBUT TABLEAU DES REGISTRES?
E325 27   E0   E307         BEQ    FONREG   OUI,RETOUR FONCTION REGISTRES
E327 108C E2A5     A        CMPY   #REGTBL+7 Y POINTE SUR PC?
E32B 27   1B   E348         BEQ    REGPC    OUI,ALLUMER CONTENU
E32D 108C E2A1     A        CMPY   #REGTBL+3 REGISTRE 8 BITS?
E331 27   0B   E33E         BEQ    REGBIT
E333 108C E2A0     A        CMPY   #REGTBL+2
E337 23   0A   E343         BLS    REBIT8
E339 30   1C       A        LEAX   -4,X     REGISTRE 16 BITS X,Y,U,P,S
E33B 16   FF7D E2BB         LBRA   TOUREG   VISUALISER CONTENU
E33E 30   1D       A REGBIT LEAX   -3,X     REGISTRE 8 BITS D,B,A,C
E340 16   FF78 E2BB         LBRA   TOUREG   VISUALISER CONTENU
E343 30   1E       A REBIT8 LEAX   -2,X     REGISTRE 8 BITS B,A,C
E345 16   FF73 E2BB         LBRA   TOUREG   VISUALISER CONTENU
E348 9E   FB       A REGPC  LDX    <SAVPIL
E34A 30   0A       A        LEAX   10,X     X POINTE SUR PC
E34C 16   FF6C E2BB         LBRA   TOUREG
```

****** PERMET DE CHANGER LE CONTENU D'UN REG LORS D'UN NMI OU SWI ******

```
E34F 31   8D FF4B   EXREGI LEAY   REGTBL,PCR
E353 9E   FB       A        LDX    <SAVPIL
E355 17   FCC8 E020         LBSR   GETKEY   ALLUMER REG
E358 81   30       A        CMPA   #$30     REG DEMANDE ?
E35A 1027 FEC9 E227         LBEQ   RPOINT   OUI,SCRUTER TOUCHES
E35E 81   06       A        CMPA   #$06     CCR?
E360 27   3B   E39D         BEQ    R8BREG   OUI
E362 30   01                INX
E364 31   21       A        LEAY   1,Y      REG SUIVT
E366 81   05       A        CMPA   #$05     ACCA?
E368 27   33   E39D         BEQ    R8BREG   OUI
E36A 30   01                INX
E36C 31   21       A        LEAY   1,Y
E36E 81   04       A        CMPA   #$04     ACCB?
E370 27   2B   E39D         BEQ    R8BREG   OUI
E372 30   01                INX
E374 31   21       A        LEAY   1,Y
E376 81   03       A        CMPA   #$03     DPR?
E378 27   23   E39D         BEQ    R8BREG   OUI
E37A 30   01                INX
E37C 31   21       A        LEAY   1,Y
E37E 81   02       A        CMPA   #$02     REG X?
E380 27   26   E3A8         BEQ    REG16B   OUI
E382 30   02       A        LEAX   2,X
E384 31   21       A        LEAY   1,Y
E386 81   01       A        CMPA   #$01     REG Y?
E388 27   1E   E3A8         BEQ    REG16B   OUI
E38A 30   02       A        LEAX   2,X
E38C 31   21       A        LEAY   1,Y
```

```
E38E 81   15        A        CMPA   #$15     REG U?
E390 27   16   E3A8          BEQ    REG16B   OUI
E392 30   02        A        LEAX   2,X
E394 31   21        A        LEAY   1,Y
E396 81   14        A        CMPA   #$14     REG PCR?
E398 27   0E   E3A8          BEQ    REG16B   OUI
E39A 16   FE8A E227          LBRA   RPOINT
E39D A6   A4        A RB8REG LDA    ,Y
E39F 97   FF        A        STA    <DISBUF+5 AFFICHE TYPE DE REG
E3A1 17   FF3E E2E2          LBSR   AF2DIG   AFFICHE CONTENU DU REG 8BITS
E3A4 30   1F                 DEX
E3A6 20   0B   E3B3          BRA    CHANG8   CHANGER LE CONTENU
E3A8 A6   A4        A REG16B LDA    ,Y
E3AA 97   FF        A        STA    <DISBUF+5 AFFICHE TYPE DE REG
E3AC 17   FF25 E2D4          LBSR   SUITEP+6 AFFICHE CONTENU DU REG 16BITS
E3AF 30   1E        A        LEAX   -2,X
E3B1 20   15   E3C8          BRA    CHAN16   CHANGER CONTENU
E3B3 34   30        A CHANG8 PSHS   Y,X      SAUVEGARDE DES POINTEURS
E3B5 8E   07FC      A        LDX    #DISBUF+2
E3B8 108E 0002      A        LDY    #$02
E3BC 17   00EF E4AE          LBSR   CHTDON   SOUS PROG DE CHGT DES DONNEES
E3BF 35   30        A        PULS   Y,X
E3C1 96   E5        A        LDA    <PRESEH
E3C3 A7   84        A        STA    ,X       NVLLE DONNEE EN PILE
E3C5 16   FF3F E307          LBRA   FONREG   SCRUTER REGISTRES

E3C8 34   30        A CHAN16 PSHS   Y,X
E3CA 8E   07FA      A        LDX    #DISBUF
E3CD 108E 0006      A        LDY    #$06
E3D1 17   00DA E4AE          LBSR   CHTDON
E3D4 35   30        A        PULS   Y,X
E3D6 DC   E5        A        LDD    <PRESEH
E3D8 ED   84        A        STD    ,X
E3DA 16   FF2A E307          LBRA   FONREG
```

EX REGI

Y pointe Deb Reg en pile

LDX Memo 07F8

GET KEY

TOUCHE REG — OUI → R POINT SAUT. — NON

A=06 — OUI → RB BREG — Reg CCR — INCX INCY

A=05 — OUI → RB BREG — Reg A — INCX INCY

X pointe valeur Reg en pile

Y pointe type Reg en code 7seg sur Reg TBL

A=04 — OUI → RB BREG — Reg B — INCX INCY

A=03 — OUI → RB DRFG — INCX INCY

A=02 → REG 16 Reg X — 2 fois INCX INCY

A=01 → REG 16 Reg Y — 2 fois INCX INCY

A=15 → REG 16 Reg U — 2 fois INCX INCY

A=14 → REG 16 Reg PC

R POINT Reset

LDA,Y
STA 6e disbuf

AF2DIG & GETKEY

DECX

CHANG 8
Empile X Y
X=3e disbuf
Y=02

CH1 DON
α,β ds 3e & 4e disbuf

desempile X Y
LDA Memo 07E5 PRESEH
STA,X

FONREG
1e → 5e disbuf
puis Redépart EXREGI

LDA,Y
STA 6e disbuf

AF4DIG & GETKEY

2 fois DECX
CHANG 16
Empile X Y
X=1e disbuf
Y=06

CH1 DON

desempile X Y
LDD Memo 07E5,E6
STD,Y
FONREG
1e → 5e disbuf
Redepart EX REGI

CHANGEMENT DU CONTENU DES REGISTRES PRESERVES EN PILE

```
E4AB 17   FB72 E020 CHTCHI LBSR  GETKEY   SCRUTER CHIFFRES
E4AE 33   8D 0000   CHTDON LEAU  INTVAL,PCR
E4B2 34   40        A      PSHS  U
E4B4 17   FC30 E0E7 MSBDON LBSR  HEXCON   CONVERTI CHIFFRE EN HEXA
E4B7 1F   89        A      TFR   A,B
E4B9 17   FC44 E100        LBSR  R7SEG
E4BC A7   80        A      STA   ,X+      ENVOI CHIFFRE SUR AFFICHEUR
E4BE 39                    RTS
E4BF 34   04        A INTVAL PSHS B
E4C1 31   3F        A      LEAY  -1,Y
E4C3 1F   20        A      TFR   Y,D      1 OU 5 OU 3
E4C5 54                    LSRB
E4C6 25   1A   E4E2        BCS   DEPLAC
E4C8 108C 0002      A SUITEP CMPY #$02    DISBUF+0
E4CC 27   0D   E4DB        BEQ   CAL16    REGISTRE 16BITS
E4CE 1F   20        A      TFR   Y,D      0 OU 4
E4D0 54                    LSRB
E4D1 24   19   E4EC        BCC   CALCU8   REGISTRE 8BITS
E4D3 108C 0000      A SUI  CMPY  #$00
E4D7 2E   D2   E4AB        BGT   CHTCHI   CHIFFRE SUIVANT
E4D9 27   06   E4E1        BEQ   RETOUR
E4DB 35   02        A CAL16 PULS A
E4DD AB   E0        A      ADDA  ,S+
E4DF 97   E6        A      STA   <PRESEL
E4E1 39              RETOUR RTS

E4E2 35   02        A DEPLAC PULS A
E4E4 48                    ASLA
E4E5 48                    ASLA
E4E6 48                    ASLA
E4E7 48                    ASLA
E4E8 34   02        A      PSHS  A
E4EA 20   DC   E4C8        BRA   SUITEP
E4EC 35   02        A CALCU8 PULS A
E4EE AB   E0        A      ADDA  ,S+
E4F0 97   E5        A      STA   <PRESEH
E4F2 20   DF   E4D3        BRA   SUI
```

CHTDON

LEAU E4BF,U+ pile

MSBDON

HEXCON α,β+ DH

A+ B

R7SEG DH+ 7seg

STA,X+

Retour si entree en MSBDON
suite si entree en CHTDON (ou CHTCHI)
empile B (val Hexa)
DECY

INTVAL Ad=E4BF

Y IMPAIR ? — oui → DEPLAC
desempile A
DH+ HD
empile A=HD

SUITEP

Y=02 — oui → CAL 16

Y PAIR ? — oui → CALCU8
desempile A
ADDA S+
STA Memo PRESEH

SUT

Y=0 — oui

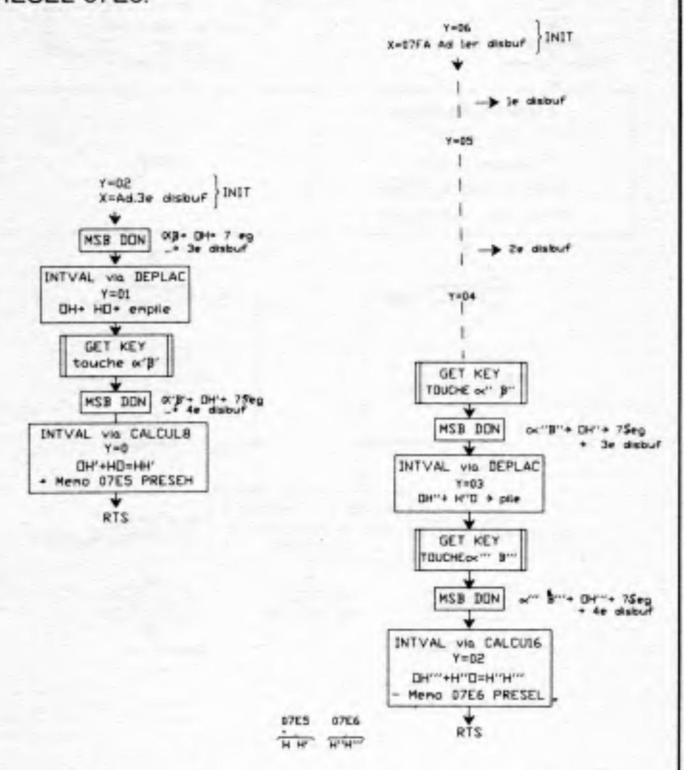
Y>0 — oui → CH CHI → GET KEY

CAL16
desempile A
ADDA,S+
STA Memo PRESEL

RTS

Les figures ci-contre représentent l'exécution de CHTDON selon l'initialisation préalable de X et de Y (Ad E385 et Ad E3CA de BX REGI) :

a) Chargement de la valeur hexadécimale HH' dans la memo PRESEH 07E5 et transcodage en 7 segments pour afficher cette valeur sur les 3e et 4e afficheurs. Ces valeurs hexa sont entrées par le clavier sous contrôle de GETKEY.
b) Même chose qu'en a) pour une valeur hexadécimale HH' H'',H'''. Dans ce dernier cas H'' H''' sont stockés en Memo PRESEL 07E6.

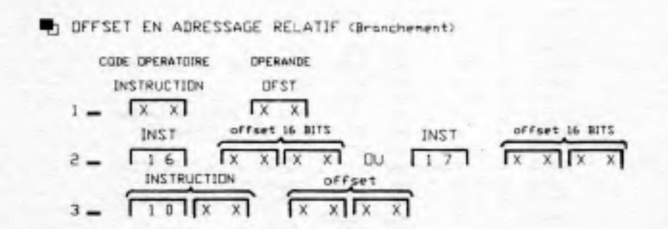


CALCUL AUTOMATIQUE D'OFFSET

Le logiciel NANOMON permet le calcul automatique dans les deux cas suivants :

- en adressage relatif (prog.EXOFS)
- en adressage indexé (prog.EXPOCT)

Nous analyserons successivement les deux programmes EXOFS et EXPOCT. Nous proposons cependant, au préalable, 2 tableaux en guise de rappels.



OFFSET EN ADRESSAGE INDEXE

1 — INST Postbyte X X X X → 5 bits d'offset & 2 bits d'indication du registre concerné

2 — INST Postbyte X X X X offset 8 ou 16 bits X X X X — Reg. concerné & longueur (8,16 bits)

3 — INSTRUCTION X X X X Postbyte X X offset X X X X

4 — CAS PARTICULIERS
label,PCR
[label,PCR]

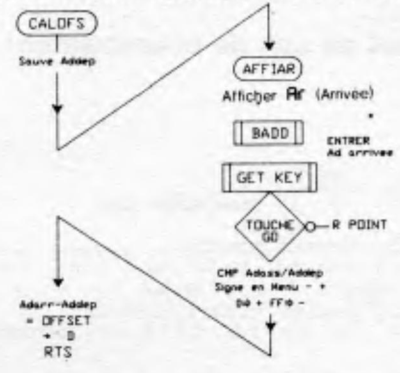
RR = 00 ≡ X
RR = 01 ≡ Y
RR = 10 ≡ U
RR = 11 ≡ S
I = 1 = Indirect
I = 0 = Direct
* = a calculer
X = indifferent

POSTBYTE		SIGNIFICATION
BINAIRE	HEXA	
0 R R * * * * *	* *	offset 5 bits
1 R R 0 0 0 0 0	* 0	INC +
1 R R I 0 0 0 1	* 1	INC ++
1 R R 0 0 0 1 0	* 2	DEC –
1 R R I 0 0 1	* 3	DEC – –
1 R R I 0 1 0 0	* 4	offset nul
1 R R I 0 1 0 1	* 5	offset AccB
1 R R I 0 1 1 0	* 6	offset AccA
1 R R I 1 0 0 0	* 8	offset 8 bits
1 R R I 1 0 0 1	* 9	offset 16 bits
1 R R I 1 0 1 1	* B	offset AccD
1 X X I 1 1 0 0	* C	offset PCR,8 bits
1 X X I 1 1 0 1	* D	offset PCR,16 bits
1 X X I 1 1 1 1	* F	Indirect entendu

EXOFST (Ad E705)

Calcul de l'Offset en cas de branchement

Ainsi que le montre l'organigramme de la page suivante, le programme EXOFST, vérifie d'abord le dernier octet entré en RAM (et éventuellement l'avant dernier octet) pour déterminer si l'offset de branchement doit comporter 8 ou 16 bits (branchement court ou branchement long). L'exécution d'OFST se poursuit ensuite par le calcul proprement dit de l'offset par appel au sous-programme CALOFST, qui calcule la différence entre l'adresse d'arrivée et l'adresse de départ, ainsi que le montre l'organigramme ci-dessous :



```
                 ****** CALCULE L'OFFSET SUR 16 BITS ******
                 **** D CONTIENT L'OFFSET,Y=ADRESSE DEP ***
                 *
                 *
                 *
E757 1F   12      A CALOFS TFR   X,Y      Y=ADRESSE DE DEPART
E759 109F EA      A        STY   <ADDDEP.
E75C 8D   05   E763        BSR   AFFIAR   AFFICHE ADRESSE ARRIVEE
E75E 1F   10      A        TFR   X,D      D=ADRESSE DEPART
E760 93   EA      A        SUBD  <ADDDEP  D=ARRIVEE-DEPART=OFFSET
E762 39                    RTS

                 ****** AFFICHAGE DE L'ADRESSE D'ARRIVEE ******
                 ** STOCKE DS NEGPOS LE SENS DE L'OFFSET(<>0) ***
                 *
                 *
                 *
E763 CC   6F41    A AFFIAR LDD   #$6F41   AFFICHE AR DS DISBUF
E766 DD   FE      A        STD   <DISBUF+4
E768 17   F94A E0B5        LBSR  BADDR    AFFICHE ADRESSE D'ARRIVEE
E76B 9F   EC      A        STX   <ADDARR
E76D 17   F8B0 E020        LBSR  GETKEY
E770 81   31      A        CMPA  #$31     TOUCHE GO?
E772 27   03   E777        BEQ   SUIVT    OUI,CHARGE ADRESSE
E774 16   FAB0 E227        LBRA  RPOINT
E777 17   FB0E E288 SUIVT  LBSR  CLRDIS   ETEINDRE AFFICHAGE
E77A 86   FF      A        LDA   #$FF
E77C 9C   EA      A        CMPX  <ADDDEP  ARRIVEE<DEPART ?
E77E 25   01   E781        BLO   STOCKA   OUI,STOCKE DS NEGPOS
E780 4C                    INCA
E781 97   E9      A STOCKA STA   <NEGPOS  NEGPOS=0>=0 NEGPOS=FF<0
E783 39                    RTS
```

Au retour de CALOFST, la Memo 07E9 contient le sens de déplacement et l'accumulateur D l'offset sur 16 bits. Le programme principal se poursuit par des tests sur le résultat de l'offset calculé notamment vérification de non dépassement de capacité, et vérification de la longueur (8 bits, 16 bits) du déplacement... Le symbole Er est affiché en cas d'erreur. Le programme se termine par le rangement de l'offset dans le prog. utilisateur et par l'affichage de cet offset dans la nouvelle adresse de départ.

```
                      ****** AFFICHAGE DE L'ADRESSE DE CHARGEMENT ******
                      ** DE L'OFFSET DANS LES QUATRE PREMIERS DIGITS ***
                      *
                      *
                      *
   / E7CB 1F   21      A AFFIAD TFR   Y,X
  |  E7CD 1F   10      A        TFR   X,D      NLLE ADRESSE DE DEP.APRES INSTRUC
  |  E7CF 17   F92E E100        LBSR  R7SEG
  |  E7D2 97   FB      A        STA   <DISBUF+1
  |  E7D4 1F   10      A        TFR   X,D
  |  E7D6 17   F923 E0FC        LBSR  L7SEG
TFRY→X→D→disbuf
  <  E7D9 97   FA      A        STA   <DISBUF
  |  E7DB 1F   98      A        TFR   B,A
  |  E7DD 17   F920 E100        LBSR  R7SEG
  |  E7E0 97   FD      A        STA   <DISBUF+3
  |  E7E2 1F   98      A        TFR   B,A
  |  E7E4 17   F915 E0FC        LBSR  L7SEG
  |  E7E7 97   FC      A        STA   <DISBUF+2
   \ E7E9 39                    RTS
```

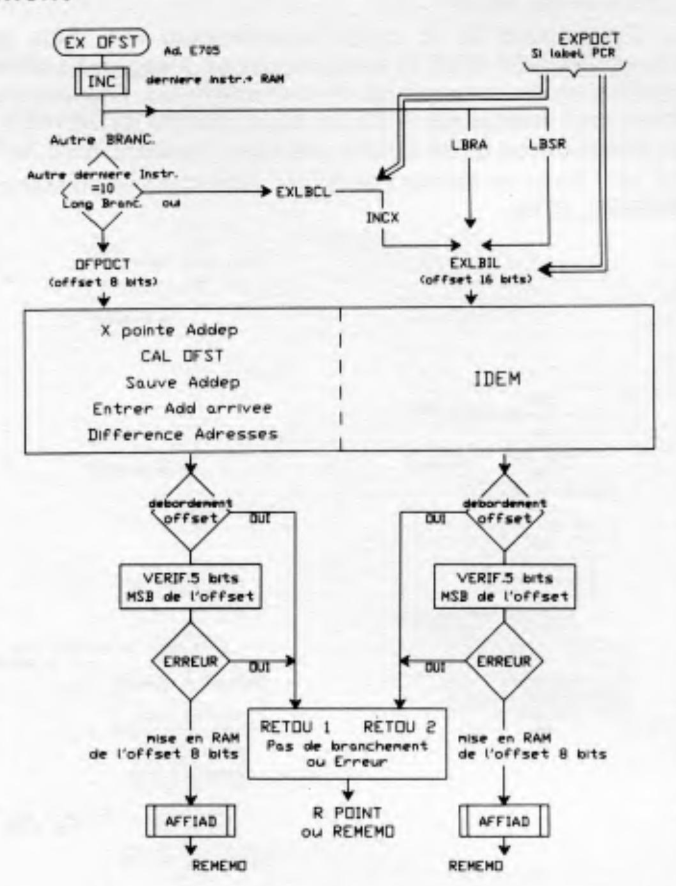
Offset en cas de branchement

```
      INST       OFST
1 - [X  X]    [X  X]   BRANCHEMENT LONG

      INST           offset
2 - [1  6]    [X  X] [X  X]  ou  [1  7]    [X  X] [X  X]  BRA. LONG

      INSTR.              offset
3 - [X  X] [X  X]    [X  X] [X  X]    BRANCHEMENT LONG
```

Organigramme général Offset de branchement

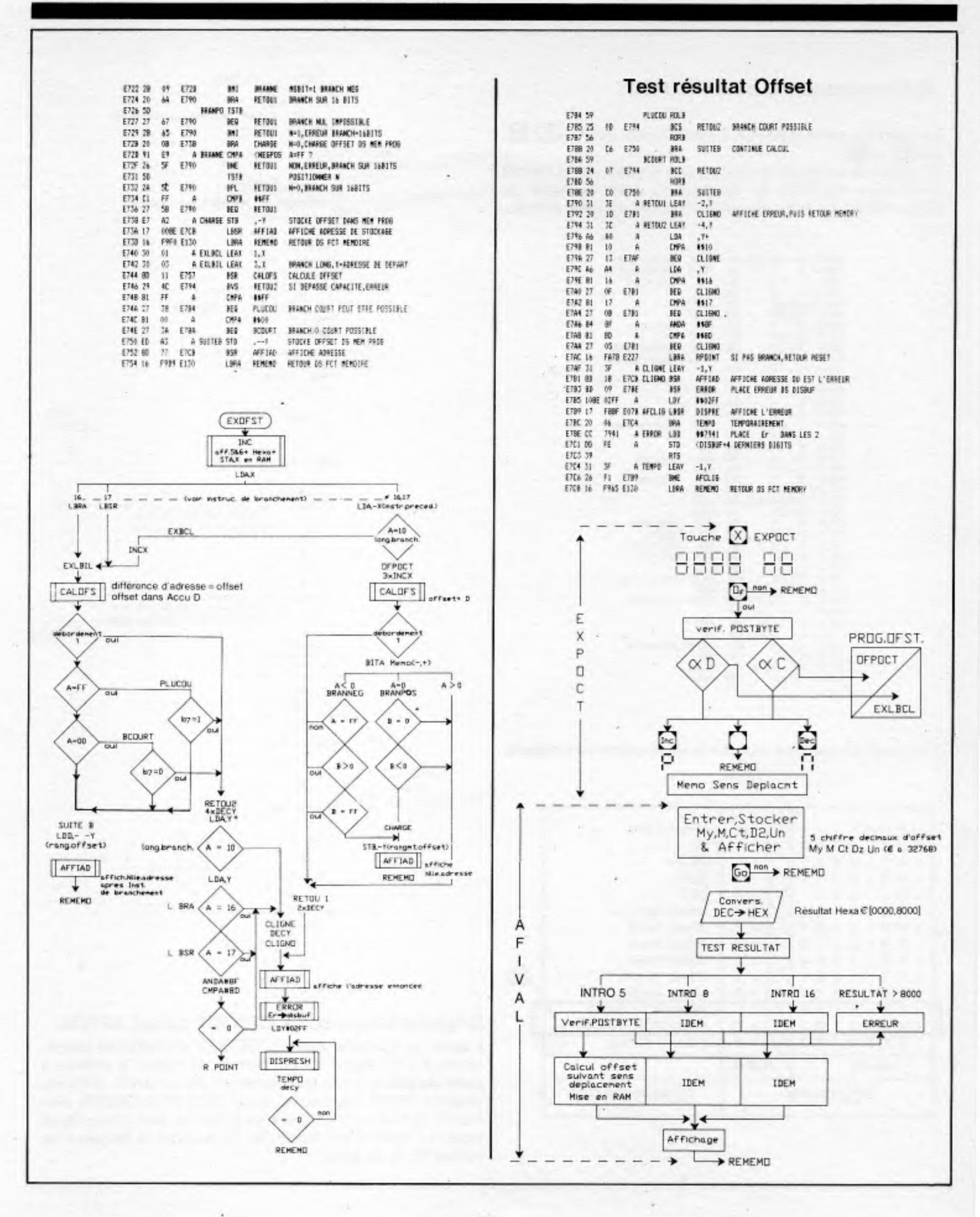


Exécution de la fonction Offset

Organigramme Exofst

EXOSFT vérifie l'instruction de branchement puis calcule l'offset en fonction de l'adresse d'arrivée désirée, laquelle est entrée par l'utilisateur par l'intermédiaire du clavier hexa. L'offset ne sera placé en RAM (dans le programme utilisateur) que si les vérifications qu'il subit s'avèrent positives.

```
E705 17   FA5B E163 EXOFST LBSR  INCREM
E708 A6   84      A        LDA   ,X
E70A 81   16      A        CMPA  #$16     LBRA?
E70C 27   34   E742        BEQ   EXLBIL   OUI,BRANCHEMANT LONG INCONDITIONNEL
E70E 81   17      A        CMPA  #$17     LBSR?
E710 27   30   E742        BEQ   EXLBIL   OUI, ////////////////////////////
E712 A6   82      A        LDA   ,-X      POINTER SUR INSTRUCTION PRECEDENTE
E714 81   10      A        CMPA  #$10     BRANCHEMENT CONDITIONNEL LONG?
E716 27   28   E740        BEQ   EXLBCL   OUI
E718 30   03      A OFPOCT LEAX  3,X      X CONTIENT ADRESSE DU BRANCH COURT
E71A 8D   3B   E757        BSR   CALOFS   CALCULER OFFSET
E71C 29   72   E790        BVS   RETOU1   SI DEPASSEMENT DE CAPACITE,ERREUR
E71E 95   E9      A        BITA  <NEGPOS
E720 27   04   E726        BEQ   BRANPO   TOUT MSB A 0 = BRANCH POS
```



```
E722 2B   09   E72D        BMI   BRANNE  MSBIT=1 BRANCH NEG
E724 20   6A   E790        BRA   RETOU1  BRANCH SUR 16 BITS
E726 5D           BRANPO TSTB
E727 27   67   E790        BEQ   RETOU1  BRANCH NUL IMPOSSIBLE
E729 2B   65   E790        BMI   RETOU1  N=1,ERREUR BRANCH=16BITS
E72B 20   0B   E738        BRA   CHARGE  N=0,CHARGE OFFSET DS MEM PROG
E72D 91   E9      A BRANNE CMPA  <NEGPOS A=FF ?
E72F 26   5F   E790        BNE   RETOU1  NON,ERREUR,BRANCH SUR 16BITS
E731 5D                    TSTB          POSITIONNER N
E732 2A   5C   E790        BPL   RETOU1  N=0,BRANCH SUR 16BITS
E734 C1   FF      A        CMPB  #$FF
E736 27   58   E790        BEQ   RETOU1
E738 E7   A2      A CHARGE STB   ,-Y     STOCKE OFFSET DANS MEM PROG
E73A 17   008E E7CB        LBSR  AFFIAD  AFFICHE ADRESSE DE STOCKAGE
E73D 16   F9F0 E130        LBRA  REMEMO  RETOUR DS FCT MEMOIRE
E740 30   01      A EXLBCL LEAX  1,X
E742 30   03      A EXLBIL LEAX  3,X     BRANCH LONG,X=ADRESSE DE DEPART
E744 8D   11   E757        BSR   CALOFS  CALCULE OFFSET
E746 29   4C   E794        BVS   RETOU2  SI DEPASSE CAPACITE,ERREUR
E748 81   FF      A        CMPA  #$FF
E74A 27   38   E784        BEQ   PLUCOU  BRANCH COURT PEUT ETRE POSSIBLE
E74C 81   00      A        CMPA  #$00
E74E 27   3A   E78A        BEQ   BCOURT  BRANCH 0 COURT POSSIBLE
E750 ED   A3      A SUITEB STD   ,--Y    STOCKE OFFSET DS MEM PROG
E752 8D   77   E7CB        BSR   AFFIAD  AFFICHE ADRESSE
E754 16   F9D9 E130        LBRA  REMEMO  RETOUR DS FCT MEMOIRE
```

Test résultat Offset

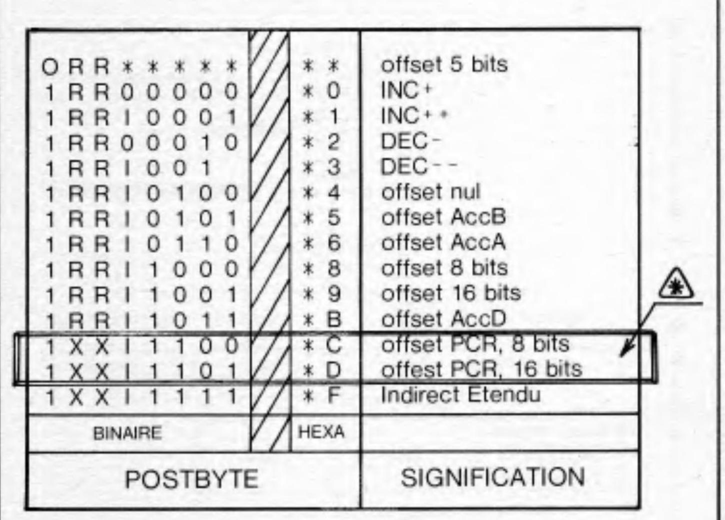
```
E784 59           PLUCOU ROLB
E785 25   0D   E794        BCS   RETOU2  BRANCH COURT POSSIBLE
E787 56                    RORB
E788 20   C6   E750        BRA   SUITEB  CONTINUE CALCUL
E78A 59           BCOURT ROLB
E78B 24   07   E794        BCC   RETOU2
E78D 56                    RORB
E78E 20   C0   E750        BRA   SUITEB
E790 31   3E      A RETOU1 LEAY  -2,Y
E792 20   1D   E7B1        BRA   CLIGNO  AFFICHE ERREUR,PUIS RETOUR MEMORY
E794 31   3C      A RETOU2 LEAY  -4,Y
E796 A6   A0      A        LDA   ,Y+
E798 81   10      A        CMPA  #$10
E79A 27   13   E7AF        BEQ   CLIGNE
E79C A6   A4      A        LDA   ,Y
E79E 81   16      A        CMPA  #$16
E7A0 27   0F   E7B1        BEQ   CLIGNO
E7A2 81   17      A        CMPA  #$17
E7A4 27   0B   E7B1        BEQ   CLIGNO
E7A6 84   BF      A        ANDA  #$BF
E7A8 81   BD      A        CMPA  #$BD
E7AA 27   05   E7B1        BEQ   CLIGNO
E7AC 16   FA78 E227        LBRA  RPOINT  SI PAS BRANCH,RETOUR RESET
E7AF 31   3F      A CLIGNE LEAY  -1,Y
E7B1 8D   18   E7CB CLIGNO BSR   AFFIAD  AFFICHE ADRESSE OU EST L'ERREUR
E7B3 8D   09   E7BE        BSR   ERROR   PLACE ERREUR DS DISBUF
E7B5 108E 02FF    A        LDY   #$02FF
E7B9 17   FBBF E07B AFCLIG LBSR  DISPRE  AFFICHE L'ERREUR
E7BC 20   06   E7C4        BRA   TEMPO   TEMPORAIREMENT
E7BE CC   7941    A ERROR  LDD   #$7941  PLACE  Er  DANS LES 2
E7C1 DD   FE      A        STD   <DISBUF+4 DERNIERS DIGITS
E7C3 39                    RTS
E7C4 31   3F      A TEMPO  LEAY  -1,Y
E7C6 26   F1   E7B9        BNE   AFCLIG
E7C8 16   F965 E130        LBRA  REMEMO  RETOUR DS FCT MEMORY
```

Organigramme début EXPOCT

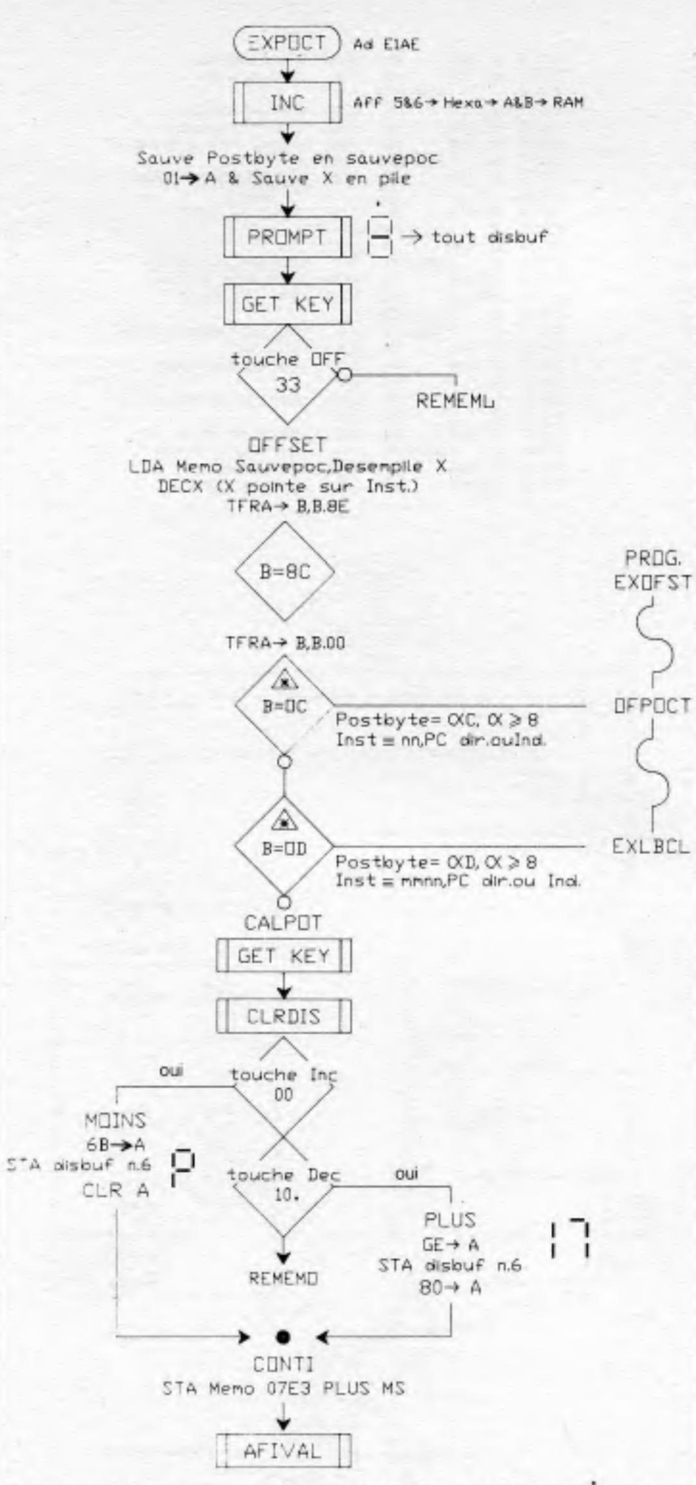
La première partie d'EXPOCT contrôle les touches X, OF, puis Inc ou Dec.
Ce début de programme vérifie aussi le postbyte et remplit la mémoire PLUSMS en fonction du sens souhaité du déplacement. La suite du programme est appelée AFIVAL.

```
E1AE 8D   83   E163 EXPOCT BSR  INCREM  POST OCTET EN MEM
E1B0 97   E4      A        STA  <SAVPOC SAUVE POST OCTET
E1B2 86   01      A        LDA  #$01    ALLUME PROMPT
E1B4 34   10      A        PSHS X
E1B6 17   00D8 E291        LBSR PROMPT
E1B9 17   FE64 E020        LBSR GETKEY
E1BC 81   33      A        CMPA #$33    TOUCHE OFFSET?
E1BE 27   03   E1C3        BEQ  OFFSET
E1C0 16   FF6D E130        LBRA REMEMO  NON,RETOUR FCT MEM
E1C3 96   E4      A OFFSET LDA  <SAVPOC REPRENDRE POST OCTET
E1C5 35   10      A        PULS X
E1C7 30   1F      A        LEAX -1,X    POINTER SUR INSTRUCTION
E1C9 1F   89      A        TFR  A,B
E1CB C4   8E      A        ANDB #$8E
E1CD C1   8C      A        CMPB #$8C    LEA N,PCR?
E1CF 26   10   E1E1        BNE  CALPOT  NON,AFFICHE SIGNE
E1D1 1F   89      A        TFR  A,B
E1D3 C4   0D      A        ANDB #$0D
E1D5 C1   0C      A        CMPB #$0C    BRANCH COURT
E1D7 1027 0530 E70B        LBEQ OFPOCT
E1DB C1   0D      A        CMPB #$0D    BRANCH LONG
E1DD 1027 055F E740        LBEQ EXLBCL
E1E1 17   FE3C E020 CALPOT LBSR GETKEY
E1E4 17   00A1 E288        LBSR CLRDIS
E1E7 81   00      A        CMPA #$00    TOUCHE INCREM?
E1E9 27   09   E1F4        BEQ  PLUS    POSTOCTET>0
E1EB 81   10      A        CMPA #$10    TOUCHE DECREM?
E1ED 27   0C   E1FB        BEQ  MOINS   POSTOCTET<0
E1EF 30   01               INX
E1F1 16   FF3C E130        LBRA REMEMO  SINON RETOUR MEM
E1F4 86   6B      A PLUS   LDA  #$6B
E1F6 97   FF      A        STA  <DISBUF+5 AFFICHE PLUS
E1F8 4F                    CLRA
E1F9 20   06   E201        BRA  CONTI
E1FB 86   6E      A MOINS  LDA  #$6E
E1FD 97   FF      A        STA  <DISBUF+5 AFFICHE MOINS
E1FF 86   80      A        LDA  #$80
E201 97   E3      A CONTI  STA  <PLUSMS
E203 16   0107 E30D        LBRA AFIVAL
```

Le tableau ci-dessous rappelle la signification du postbyte.



POSTBYTE (BINAIRE)	HEXA	SIGNIFICATION
O R R * * * * *	* *	offset 5 bits
1 R R 0 0 0 0 0	* 0	INC +
1 R R I 0 0 0 1	* 1	INC + +
1 R R 0 0 0 1 0	* 2	DEC -
1 R R I 0 0 1	* 3	DEC - -
1 R R I 0 1 0 0	* 4	offset nul
1 R R I 0 1 0 1	* 5	offset AccB
1 R R I 0 1 1 0	* 6	offset AccA
1 R R I 1 0 0 0	* 8	offset 8 bits
1 R R I 1 0 0 1	* 9	offset 16 bits
1 R R I 1 0 1 1	* B	offset AccD
1 X X I 1 1 0 0	* C	offset PCR, 8 bits
1 X X I 1 1 0 1	* D	offest PCR, 16 bits
1 X X I 1 1 1 1	* F	Indirect Etendu



Organigramme suite EXPOCT début AFIVAL

L'appel en cascade des s/p GETKEY et CHTDON (initialisation X = 1er disbuf Y = 06) permet de rentrer 4 chiffres à partir du clavier et de les classer en Memo 07E5, 07E6 (cf. chapitre RSWI). Un nouvel appel GETKEY/MSBDON permet de rentrer un 5e chiffre. Ces 5 chiffres sont convertis en hexa. Le résultat est testé afin de mesurer la longueur de l'offset (5, 8, 16 bits).

```
E3DD 30    01           AFIVAL INX
E3DF 34    10        A         PSHS   X
E3E1 17    FC3C E020           LBSR   GETKEY
E3E4 8E    07FA      A         LDX    #DISBUF
E3E7 108E  0006      A         LDY    #$06
E3EB 17    00C0 E4AE           LBSR   CHTDON
E3EE 17    FC2F E020           LBSR   GETKEY
E3F1 17    00C0 E4B4           LBSR   MSBDON
E3F4 0F    E7        A         CLR    <PRELOW
E3F6 D7    E8        A         STB    <PRELOW+1
E3F8 35    10        A         PULS   X
E3FA 17    FC23 E020           LBSR   GETKEY
E3FD 81    31        A         CMPA   #$31
E3FF 27    03   E404           BEQ    CALCON
E401 16    FD2C E130           LBRA   REMEMO
E404 17    FEB1 E2B8 CALCON    LBSR   CLRDIS
E407 17    00EA E4F4           LBSR   DECHEX   CONVERSION DEC-HEXA
E40A DC    E7        A         LDD    <PRELOW
E40C 4D                        TSTA            A=0?
E40D 26    0B   E41A           BNE    TESTST   NON,VAL 16BITS?
E40F 58                        ASLB
E410 25    18   E42B           BCS    TSTSUI   VAL 16BITS?
E412 58                        ASLB
E413 25    52   E467           BCS    INTRO8   VAL 8BITS?
E415 58                        ASLB
E416 25    4F   E467           BCS    INTRO8   VAL 8BITS
E418 20    1A   E434           BRA    INTRO5   VAL 5BITS
E41A 81    80        A TESTST  CMPA   #$80     A>80?
E41C 22    08   E426           BHI    ERREUR   OUI,AFFICHE ERREUR
E41E 27    02   E422           BEQ    COMPAB
E420 20    66   E488           BRA    INTR16   NON,VAL 16BITS
E422 C1    00        A COMPAB  CMPB   #$00
E424 27    62   E488           BEQ    INTR16
E426 30    1F        A ERREUR  LEAX   -1,X
E428 1F    12        A EREURE  TFR    X,Y
E42A 16    0384 E7B1           LBRA   CLIGNO
E42B 56                TSTSUI  RORB
E42E C1    80        A         CMPB   #$80     B>80?
E430 22    56   E488           BHI    INTR16   OUI,VAL 16BITS
E432 20    33   E467           BRA    INTRO8   NON,VAL 8BITS
```

AFIVAL E3DD

INC X (X pointe postbyte)
empile X

GET KEY

X=1er disbuf Y=D6

CHT DON — 07E5 MY M — 07E6 Ct D_z

GET KEY

MSBDON — DUn → B

ENTRER 5 CHIFFRES DECIMAUX ∈ [0, 32768]

CLR PRELOW
STB PRELOW+1
Desempile X — 07E7 0 0 — 07E8 0 Un

GET KEY

touche [0] — REMEMO

CALCON

CLRDIS

DEC→ HEXA
Un
+DA × Dz
+64 × Ct
+3E8 × M
+2710 × My
Σ → Memo 07E7,8
My > 3 → ERREUR

LDD PRELOW (MEMO 07E7)

TEST A≠0 — A=0

A > 80 — A=80 B≠0 — A < 80 B > 80 — B=80 — B < 7F

B ≥ 40 — B < 3F

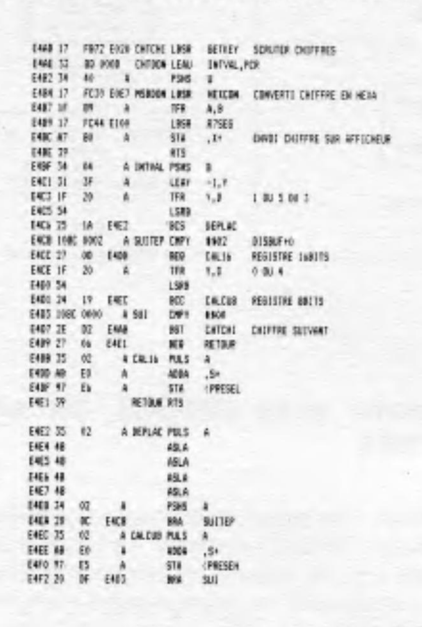
B ≥ 20 — B < 1F

ERREUR
Σ > (8000)hexa
DECX
TFRX→ Y
Er

INTRO 16 — INTRO 8 — INTRO 5

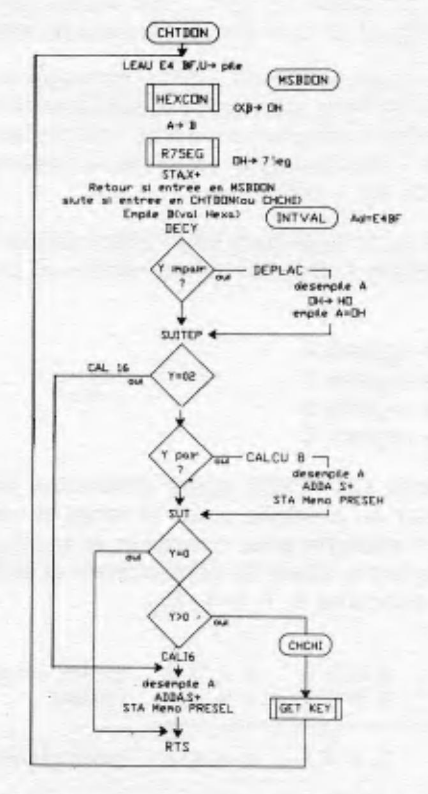
Chargement des mémoires 07E5 07E6 par MyM C + D_z converti Hexa H H' H'' H'''

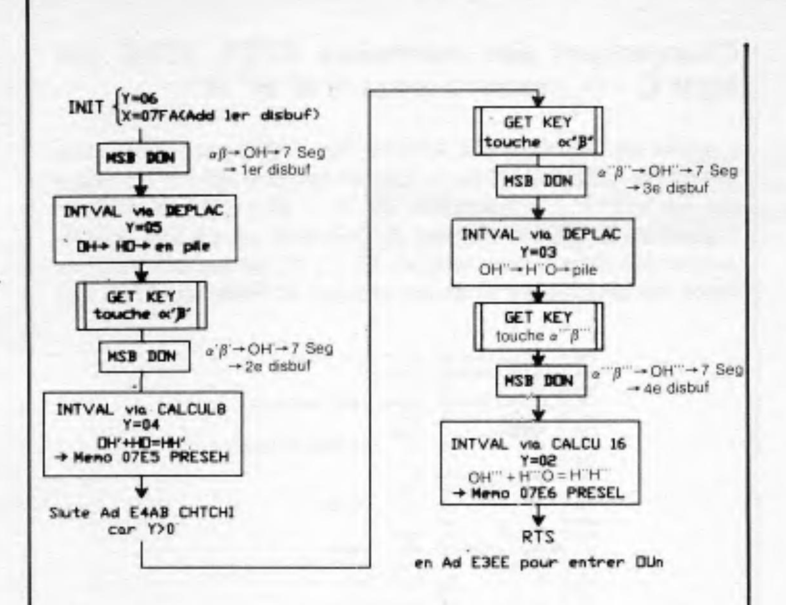
L'appel de GETKEY par AFIVAL (Ad E3E1) permet d'entrer les dz de milliers (My) par le clavier (touche $\alpha\beta \rightarrow$ A au retour de GETKEY). L'initialisation de X Y (X = 07FA Y = 06) et l'appel de CHTDON permet de convertir $\alpha\beta$ en OH puis de rentrer les trois autres valeurs M, C_t, D_z de les convertir en hexa, de les afficher et de les stocker en RAM (en 07E5,E6).



```
E4A8 17    FB72 E020 CHTCHI LBSR  GETKEY   SCRUTER CHIFFRES
E4AE 33    8D 000B   CHTDON LEAU  INTVAL,PCR
E4B2 34    40        A      PSHS  U
E4B4 17    FC30 E0E7 MSBDON LBSR  HEXCON   CONVERTI CHIFFRE EN HEXA
E4B7 1F    89        A      TFR   A,B
E4B9 17    FC44 E100        LBSR  R7SEG
E4BC A7    80        A      STA   ,X+      ENVOI CHIFFRE SUR AFFICHEUR
E4BE 39                     RTS
E4BF 34    04        A INTVAL PSHS B
E4C1 31    3F        A      LEAY  -1,Y
E4C3 1F    20        A      TFR   Y,D      1 OU 5 OU 3
E4C5 54                     LSRB
E4C6 25    1A   E4E2        BCS   DEPLAC
E4C8 108C  0002      A SUITEP CMPY #$02    DISBUF+0
E4CC 27    0D   E4DB        BEQ   CAL16    REGISTRE 16BITS
E4CE 1F    20        A      TFR   Y,D      0 OU 4
E4D0 54                     LSRB
E4D1 24    19   E4EC        BCC   CALCU8   REGISTRE 8BITS
E4D3 108C  0000      A SUI  CMPY  #$00
E4D7 2E    D2   E4AB        BGT   CHTCHI   CHIFFRE SUIVANT
E4D9 27    06   E4E1        BEQ   RETOUR
E4DB 35    02        A CAL16 PULS A
E4DD AB    E0        A      ADDA  ,S+
E4DF 97    E6        A      STA   <PRESEL
E4E1 39              RETOUR RTS

E4E2 35    02        A DEPLAC PULS A
E4E4 48                     ASLA
E4E5 48                     ASLA
E4E6 48                     ASLA
E4E7 48                     ASLA
E4E8 34    02        A      PSHS  A
E4EA 20    DC   E4C8        BRA   SUITEP
E4EC 35    02        A CALCU8 PULS A
E4EE AB    E0        A      ADDA  ,S+
E4F0 97    E5        A      STA   <PRESEH
E4F2 20    DF   E4D3        BRA   SUI
```





Organigramme suite EXPOCT, fin AFIVAL - Partie INTRO5

Les cinq chiffres décimaux du déplacement entrés à partir du clavier ont été convertis en Hexa. Le résultat Hexa subit différents tests afin de mesurer la longueur (5, 8 ou 16 bits) de l'offset. En particulier, un déplacement compris entre $[0, +15]_{dec}$, c'est-à-dire $[0, OF]_{hexa}$ est traduit par un offset codé sur 5 bits et ce quel que soit le sens de déplacement.

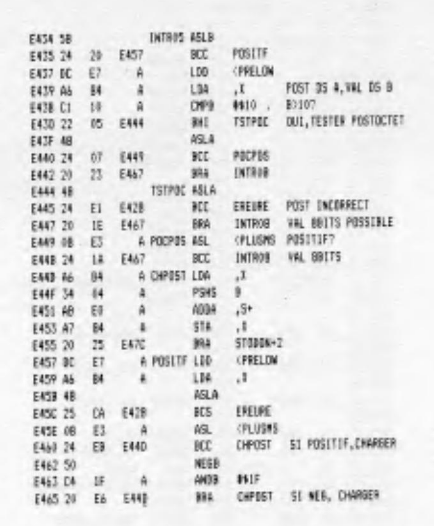
Si le déplacement est positif, l'offset sera égal au déplacement transcodé hexa alors que si le déplacement est négatif, l'offset sera le complément à 2 du nombre hexa. Notons encore qu'à 1 déplacement négatif de 16 correspond aussi 1 offset codé sur 5 bits.

Dans le cas du codage sur 5 bits, l'offset devant être inclus dans le postbyte, l'utilisateur devra «écrire» un postbyte initial :

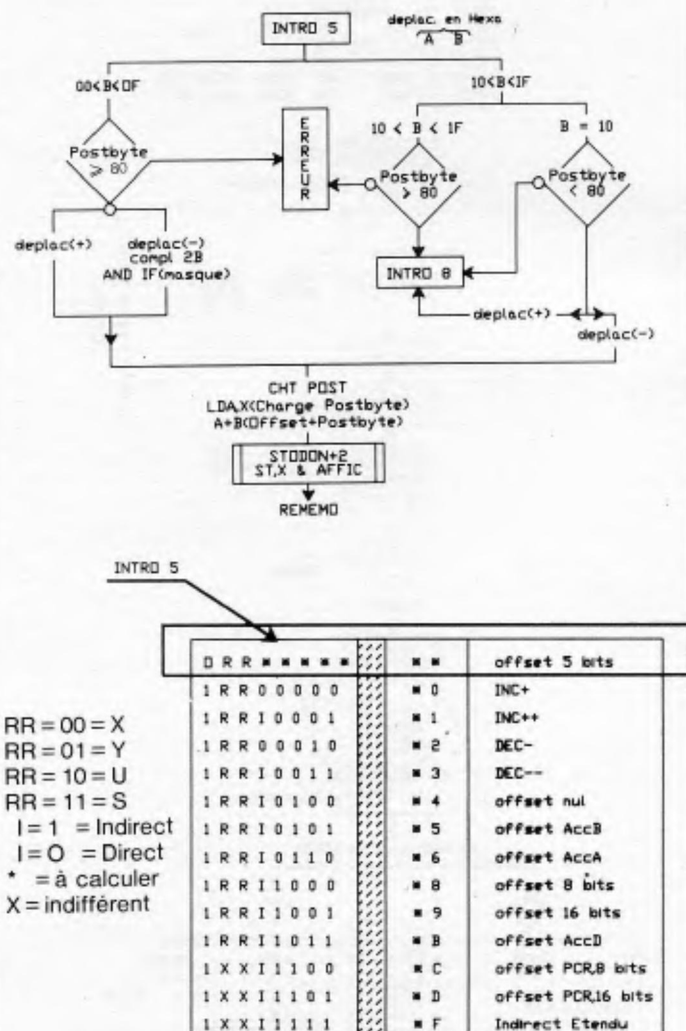
- 00 pour le registre X
- 20 pour le registre Y
- 40 pour le registre U
- 60 pour le registre S

INTR05 (partie CH POST) après différentes vérifications ajoute l'offset au postbyte initial et range le résultat à la place de ce postbyte pour constituer le postbyte définitif incluant à la fois la valeur du déplacement et la désignation du registre concerné X, Y, U ou S.

```
    0 R R 0    0 0 0 0    (postb. initial)
+   0 0 0 *    * * * *    (offset)
---------------------------
    0 * * *    * * * *    (postbyte final)
```



```
E434 58             INTR05 ASLB
E435 24   20  E457         BCC    POSITF
E437 DC   E7     A         LDD    <PRELOW
E439 A6   84     A         LDA    ,X       POST DS A,VAL DS B
E43B C1   10     A         CMPB   #$10  .  B>10?
E43D 22   05  E444         BHI    TSTPOC   OUI,TESTER POSTOCTET
E43F 48                    ASLA
E440 24   07  E449         BCC    POCPOS
E442 20   23  E467         BRA    INTR08
E444 48             TSTPOC ASLA
E445 24   E1  E428         BCC    EREURE   POST INCORRECT
E447 20   1E  E467         BRA    INTR08   VAL 8BITS POSSIBLE
E449 08   E3     A  POCPOS ASL    <PLUSMS  POSITIF?
E44B 24   1A  E467         BCC    INTR08   VAL 8BITS
E44D A6   84     A  CHPOST LDA    ,X
E44F 34   04     A         PSHS   B
E451 AB   E0     A         ADDA   ,S+
E453 A7   84     A         STA    ,X
E455 20   25  E47C         BRA    STODON+2
E457 DC   E7     A  POSITF LDD    <PRELOW
E459 A6   84     A         LDA    ,X
E45B 48                    ASLA
E45C 25   CA  E428         BCS    EREURE
E45E 08   E3     A         ASL    <PLUSMS
E460 24   EB  E44D         BCC    CHPOST   SI POSITIF,CHARGER
E462 50                    NEGB
E463 C4   1F     A         ANDB   #$1F
E465 20   E6  E44D         BRA    CHPOST   SI NEG, CHARGER
```

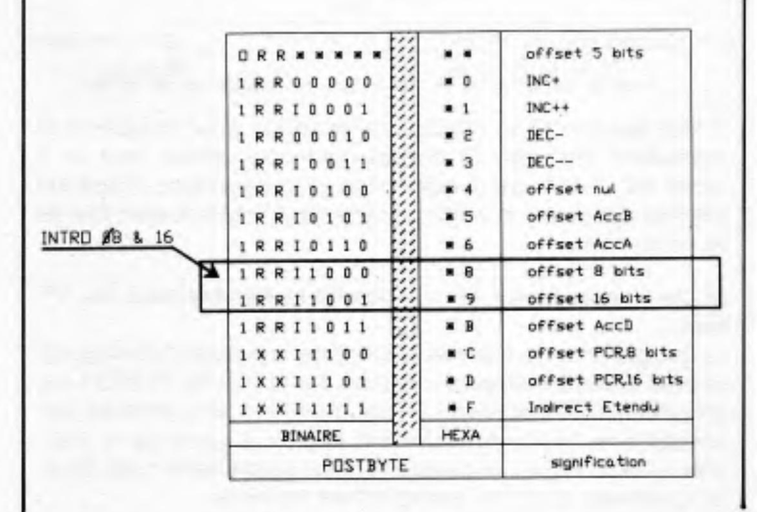


POSTBYTE BINAIRE	POSTBYTE HEXA	signification
0 R R * * * * *	* *	offset 5 bits
1 R R 0 0 0 0 0	* 0	INC+
1 R R I 0 0 0 1	* 1	INC++
1 R R 0 0 0 1 0	* 2	DEC-
1 R R I 0 0 1 1	* 3	DEC--
1 R R I 0 1 0 0	* 4	offset nul
1 R R I 0 1 0 1	* 5	offset AccB
1 R R I 0 1 1 0	* 6	offset AccA
1 R R I 1 0 0 0	* 8	offset 8 bits
1 R R I 1 0 0 1	* 9	offset 16 bits
1 R R I 1 0 1 1	* B	offset AccD
1 X X I 1 1 0 0	* C	offset PCR,8 bits
1 X X I 1 1 0 1	* D	offset PCR,16 bits
1 X X I 1 1 1 1	* F	Indirect Etendu

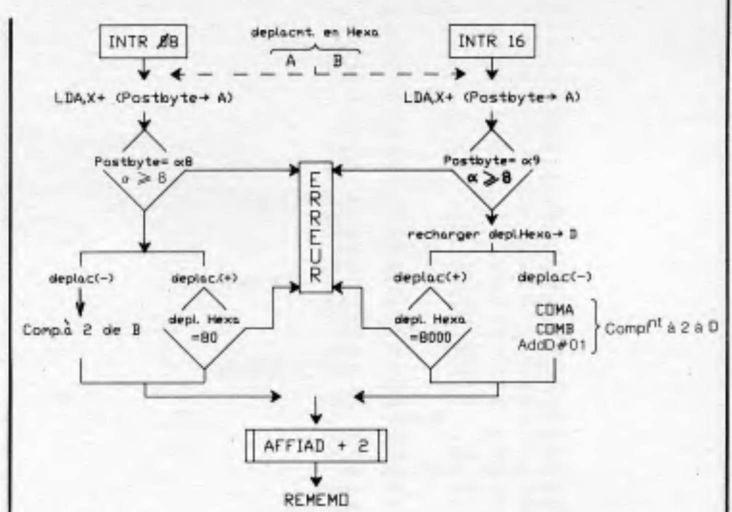
Organigramme suite EXPOCT fin AFIVAL Parties INTRO8 et INTRO16

Si le déplacement appartient à l'intervalle [– 128, + 127] AFIVAL effectue un branchement à INTR 08 alors que pour un déplacement∈[– 32768, + 32767], c'est à INTR16 que s'effectue le branchement. INTR08 et INTR16 calculent respectivement l'offset sur 8 ou 16 bits à partir du déplacement entrée sur 5 chiffres décimaux à partir du clavier et convertion en Hexa. L'offset est rangé en RAM dans le programme utilisateur immédiatement après le postbyte. Offset et adresse correspondante sont également affichés grâce à AFFIAD + 2.

```
E467 DC   E7        A INTR08 LDD   <PRELOW
E469 A6   80        A        LDA   ,X+
E46B 49                      ROLA
E46C 24   B8   E426          BCC   ERREUR   SI DIFFERENT 8BITS,ERREUR
E46E 46                      RORA
E46F 84   0F        A        ANDA  #$0F
E471 81   08        A        CMPA  #$08     POST OCTET 8BITS?
E473 26   B1   E426          BNE   ERREUR
E475 08   E5        A        ASL   <PLUSMS  OUI,POSITIF?
E477 24   09   E482          BCC   STADON   OUI,STOCKER DONNEE
E479 50                      NEGB
E47A E7   84        A STODON STB   ,X
E47C 17   034E E7CD          LBSR  AFFIAD+2
E47F 16   FCAE E130          LBRA  REMEMO
E482 C1   80        A STADON CMPB  #$80
E484 27   A0   E426 RCR      BEQ   ERREUR
E486 20   F2   E47A          BRA   STODON
E488 A6   80        A INTR16 LDA   ,X+
E48A 49                      ROLA
E48B 24   99   E426          BCC   ERREUR   DIFF 16BITS
E48D 46                      RORA
E48E 84   0F        A        ANDA  #$0F
E490 81   09        A        CMPA  #$09     16BITS?
E492 26   92   E426          BNE   ERREUR
E494 DC   E7        A        LDD   <PRELOW
E496 08   E5        A        ASL   <PLUSMS  POSITIF?
E498 24   09   E4A3          BCC   COMPAR   OUI
E49A 43                      COMA
E49B 53                      COMB
E49C C3   0001      A        ADDD  #$01
E49F ED   84        A CHAR16 STD   ,X
E4A1 20   D9   E47C          BRA   STODON+2
E4A3 1083 8000      A COMPAR CMPD  #$8000
E4A7 27   DB   E484          BEQ   RCR
E4A9 20   F4   E49F          BRA   CHAR16
```

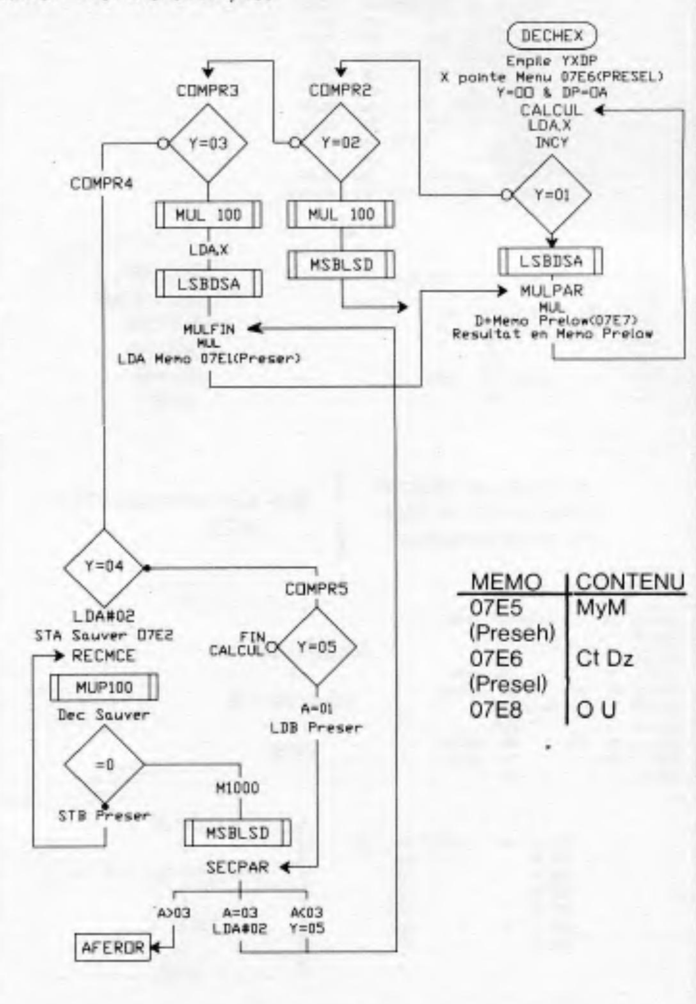


POSTBYTE BINAIRE	POSTBYTE HEXA	signification
0 R R * * * * *	* *	offset 5 bits
1 R R 0 0 0 0 0	* 0	INC+
1 R R I 0 0 0 1	* 1	INC++
1 R R 0 0 0 1 0	* 2	DEC-
1 R R I 0 0 1 1	* 3	DEC--
1 R R I 0 1 0 0	* 4	offset nul
1 R R I 0 1 0 1	* 5	offset AccB
1 R R I 0 1 1 0	* 6	offset AccA
1 R R I 1 0 0 0	* 8	offset 8 bits
1 R R I 1 0 0 1	* 9	offset 16 bits
1 R R I 1 0 1 1	* B	offset AccD
1 X X I 1 1 0 0	* C	offset PCR,8 bits
1 X X I 1 1 0 1	* D	offset PCR,16 bits
1 X X I 1 1 1 1	* F	Indirect Etendu



Conversion DEC→ HEXA

Ce sous-programme de conversion appelé par AFIVAL convertit le déplacement My MCtDU, en Hexa et place le résultat en Memos 07E7,E8.



MEMO	CONTENU
07E5 (Preseh)	MyM
07E6 (Presel)	Ct Dz
07E8	O U

```
E4F4 34   38        A DECHEX PSHS   Y,X,DP
E4F6 86   0A        A        LDA    #$0A
E4F8 1F   8B        A        TFR    A,DP   10 DS DP
E4FA 8E   07E6      A        LDX    #PRESEL
E4FD 108E 0000      A        LDY    #$00
E501 A6   84        A CALCUL LDA    ,X
E503 31   21        A        LEAY   1,Y
E505 108C 0001      A        CMPY   #$01
E509 26   0B   E516          BNE    COMPR2
E50B 8D   66   E573          BSR    LSBOSA  LSB DANS ACCA
E50D 3D                MULPAR MUL
E50E F3   07E7      A        ADDD   PRELOW  D+PRELOW DS D
E511 FD   07E7      A        STD    PRELOW
E514 20   EB   E501          BRA    CALCUL
E516 108C 0002      A COMPR2 CMPY   #$02
E51A 26   06   E522          BNE    COMPR3
E51C 8D   6D   E58B          BSR    MUL100  MUL DES CENTAINES
E51E 8D   62   E582          BSR    MSBLSB
E520 20   EB   E50D          BRA    MULPAR
E522 108C 0003      A COMPR3 CMPY   #$03
E526 26   0C   E534          BNE    COMPR4
E528 8D   61   E58B          BSR    MUL100  MUL DES MILLIERS
E52A A6   84        A        LDA    ,X
E52C 8D   45   E573          BSR    LSBOSA
E52E 3D                MULFIN MUL
E52F B6   07E1      A        LDA    PRESER
E532 20   D9   E50D          BRA    MULPAR
E534 108C 0004      A COMPR4 CMPY   #$04
E538 26   23   E55D          BNE    COMPR5
E53A 86   02        A        LDA    #$02
E53C B7   07E2      A        STA    SAUVER
E53F 8D   50   E591 RECMCE BSR    MUP100  MUL DES DIZAINES DE MILLIERS
E541 7A   07E2      A        DEC    SAUVER
E544 27   05   E54B          BEQ    M10000
E546 F7   07E1      A        STB    PRESER
E549 20   F4   E53F          BRA    RECMCE
E54B 8D   35   E582 M10000 BSR    MSBLSB
E54D 81   03        A SECPAR CMPA   #$03
E54F 24   06   E557          BHS    TESTER
E551 108E 0005      A        LDY    #$05
E555 20   D7   E52E          BRA    MULFIN
E557 22   11   E56A TESTER BHI    AFEROR
E559 86   02        A        LDA    #$02
E55B 20   D1   E52E          BRA    MULFIN
E55D 108C 0005      A COMPR5 CMPY   #$05
E561 26   0E   E571          BNE    FINCAL
E563 86   01        A        LDA    #$01
E565 F6   07E1      A        LDB    PRESER
E568 20   E3   E54D          BRA    SECPAR
E56A 35   78        A AFEROR PULS   DP,X,Y,U
E56C 1F   12        A        TFR    X,Y
E56E 16   0240 E7B1          LBRA   CLIEND
E571 35   B8        A FINCAL PULS   PC,X,Y,DP
```

```
E58B 33   8D 0008    MUL100 LEAU   INTER3,PCR
E58F 34   40        A        PSHS   U
E591 1F   B9        A MUP100 TFR    DP,B
E593 1F   98        A        TFR    B,A
E595 3D                      MUL
E596 39                      RTS
E597 F7   07E1      A INTER3 STB    PRESER
E59A 39                      RTS
```

MUL100
PC pile = E597
MUP100
OAXOA
64→B
RTS*

*Si entrée en MUL100 branchement en E597 lors du désempilage } STB Memo Preser 07E1 RTS

```
E573 C6   08        A LSBOSA LDB    #$08
E575 48                DECENC LSLA
E576 5A                      DECB
E577 C1   04        A        CMPB   #$04
E579 26   FA   E575          BNE    DECENC
E57B 44                DECTJS LSRA
E57C 5A                      DECB
E57D 26   FC   E57B          BNE    DECTJS
E57F 1F   B9        A        TFR    DP,B
E581 39                      RTS
```

$A = \alpha\beta \rightarrow O\beta = A$
↓
$DP = OA \rightarrow B$
↓
RTS

```
E582 A6   84        A MSBLSB LDA    ,X
E584 44                      LSRA
E585 44                      LSRA
E586 44                      LSRA
E587 44                      LSRA
E588 30   1F        A        LEAX   -1,X
E58A 39                      RTS
```

LDA,X
↓
$A = \alpha\beta \rightarrow O\alpha = A$
↓
DEC X
↓
RTS

Interface avec un magnétophone à cassette

Généralités

L'interface avec un magnéto à cassette qui se trouve sur la partie clavier/visualisation donne à l'utilisateur la possibilité de sauver et de relire ses programmes sur un magnétophone ordinaire. Le standard d'enregistrement utilise une modulation de durée (P.D.M.). L'enregistrement a les caractéristiques suivantes :

1) Un «1» logique est traduit par une période de signal rectangulaire de # 360 μs

"1" — 360 μs

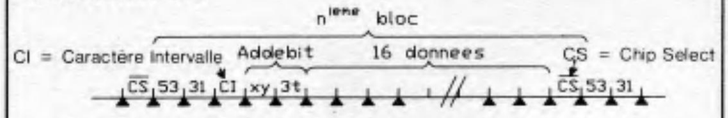
2) Un «0» logique est traduit par une période de signal rectangulaire de # 1440 μs. [La durée d'un «0» est quatre fois plus importante que celle d'un «1»].

"0" — 1440 μs

3) Un octet est enregistré sous la forme d'un «0» suivi des 8 bits significatifs de l'octet proprement dit et d'un «1» d'identification de fin.

4) Les données du programme à sauvegarder sont regroupées par 16 octets dans des blocs consécutifs de 22 caractères. Les 1er et dernier blocs peuvent comporter moins de 16 octets de données. En effet, si le programme RAM à transférer commence par exemple à l'adresse 0105, le 1er bloc comportera les données d'adresse 0105 à 010F, le 2e bloc les données d'adresse 0110 à 011F, etc

5) La structure d'un bloc est la suivante : Caractère 53, caractère 31, caractère intervalle, MSB Adresse début bloc, LSB, 16 datas, caractère de checksum. Pour le dernier bloc, le caractère 31 est remplacé par le caractère 4A.



Il faut ajouter 03 au nombre de données pour composer le caractère intervalle (2 octets d'adresse début bloc et 1 octet de checksun). L'adjonction d'un caractère checksun permet de vérifier la bonne qualité de la transmission lors de la réception.

6) Deux caractères FF précèdent la transmission du 1er bloc.
Le programme de transfert [données → magnétophone] est appelé lorsque l'utilisateur appuie sur la touche PUNCH. Le programme de transfert inverse [données enregistrées sur cassette → Mémoire RAM] est appelé à partir de la touche LOAD. Nous analyserons successivement ces deux programmes dans les paragraphes suivants.

Chargement d'un programme d'une zone mémoire vers un magnétocassette

```
E641 8E    003D    A EXPUNC LDX   #$003D
E644 9F    FE      A        STX   <DISBUF+4 AFFICHER S POUR START
E646 17    FA6C E0B5        LBSR  BADDR    FABRIQUER ADRESSE DE DEBUT
E649 9F    F2      A        STX   <SADDR1  SAUVE DEBUT ADRESSE
E64B 86    69      A        LDA   #$69     AFFICHER F POUR FIN
E64D 97    FF      A        STA   <DISBUF+5
E64F 17    FA63 E0B5        LBSR  BADDR    FABRIQUER ADRESSE DE FIN
E652 9F    F4      A        STX   <SADDR2  SAUVE FIN ADRESSE
E654 4F                     CLRA           ACCES A DDRB
E655 B7    A007    A        STA   SCNCNT
E658 4A                     DECA           PB EN SORTIE
E659 B7    A005    A        STA   DISCNT
E65C 86    04      A        LDA   #$04     ACCES A ORB DU PIA
E65E B7    A007    A        STA   SCNCNT
E661 4F                     CLRA           LIGNE No 0 A 0 ET PB6 = 0
E662 B7    A005    A        STA   DISCNT
E665 86    FF      A        LDA   #$FF
E667 8D    71   E6DA        BSR   DLY5MS   FABRICATION D'UNE SERIE
E669 8D    6F   E6DA        BSR   DLY5MS   DE 10 PERIODES DE 1400 ET 5600HZ
E66B 8D    63   E6D0 RECYCL BSR   DEBCHA   CHARGER CARAC DE DEBUT
E66D 86    FF      A        LDA   #$FF
E66F D6    F3      A        LDB   <SADDR1+1 POIDS FAIBLE ADRESSE DEBUT
E671 34    04      A        PSHS  B
E673 A0    E0      A        SUBA  ,S+      A - B DANS ACCA
E675 84    0F      A CODFIN ANDA  #$0F
E677 8B    03      A        ADDA  #$03     TRANSMISSION D'UN CARACTERE
E679 8D    5F   E6DA        BSR   DLY5MS   INTERVALE
E67B 97    F1      A        STA   <SAVREG
E67D 8E    07F2    A        LDX   #SADDR1  DEBUT ADRESSE
E680 8D    56   E6D8        BSR   TRANSM   TRANSMISSION MSB ADRESSE DEBUT
E682 9B    F1      A        ADDA  <SAVREG
E684 97    F1      A        STA   <SAVREG
E686 30    01      A        LEAX  1,X
E688 8D    4E   E6D8        BSR   TRANSM   TRANSMISSION LSB ADRESSE DEBUT
E68A 9B    F1      A        ADDA  <SAVREG
E68C 97    F1      A        STA   <SAVREG
E68E 9E    F2      A        LDX   <SADDR1  X POINTE SUR PROGRAMME A ENREGISTRER
E690 8D    46   E6D8 TJTRAN BSR   TRANSM   TRANSMISSION DES DONNEES
E692 9B    F1      A        ADDA  <SAVREG
E694 97    F1      A        STA   <SAVREG
E696 9C    F4      A        CMPX  <SADDR2  FIN D'ENREGISTREMENT ?
E698 27    0A   E6A4        BEQ   FINENR   OUI,TRANSMETTRE FIN
E69A 30    01      A        LEAX  1,X      NON,DONNEE SUIVANTE A TRANSMETTRE
E69C 9F    F2      A        STX   <SADDR1  SAUVE POINTEUR
E69E 96    F3      A        LDA   <SADDR1+1
E6A0 85    0F      A        BITA  #$0F     16 CARACTERES TRANSMIS ?
E6A2 26    EC   E690        BNE   TJTRAN   NON,CONTINUER TRANSMISSION
E6A4 96    F1      A FINENR LDA   <SAVREG  OUI,16 CARACTERES TRANSMIS
E6A6 40                     NEGA
E6A7 8D    31   E6DA        BSR   DLY5MS   TRANSMISSION CODE POUR 16CARAC TRANSMIS
E6A9 9C    F4      A        CMPX  <SADDR2  FIN PROGRAMME ?
E6AB 27    18   E6C5        BEQ   FINCHA   OUI,TRANSMETTRE FIN CHARGEMENT
E6AD 96    F2      A        LDA   <SADDR1  NON,MSB DEBUT ADRESSE
E6AF 91    F4      A        CMPA  <SADDR2  < MSB FIN ADRESSE ?
E6B1 2B    B8   E66B        BMI   RECYCL   OUI,RECOMMENCER CYCLE
E6B3 D6    F3      A        LDB   <SADDR1+1 NON,LSB DEBUT ADRESSE
E6B5 96    F5      A        LDA   <SADDR2+1 - LSB FIN ADRESSE <> 0 ?
E6B7 84    F0      A        ANDA  #$F0
E6B9 34    04      A        PSHS  B
E6BB A1    E0      A        CMPA  ,S+      COMPARE B A ACCA
E6BD 26    AC   E66B        BNE   RECYCL   OUI,RECOMMENCER UN CYCLE
E6BF 8D    0F   E6D0        BSR   DEBCHA   NON,TRANSMETTRE DEBUT CHARGEMENT
E6C1 96    F5      A        LDA   <SADDR2+1 DU CODE DE FIN
E6C3 20    B0   E675        BRA   CODFIN   CHARGEMENT PUIS FIN
```

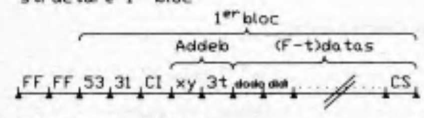
- $\alpha.\beta \rightarrow$ T

  Signifie transmission de $\alpha\beta$

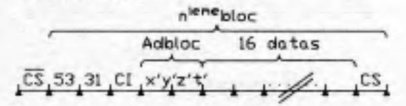
- $\alpha\beta \rightarrow$ T*

  Signifie transmission de $\alpha\beta$ et addition de $\alpha\beta$ a Memo 07F1 pour elaborer checksum

- structure 1er bloc



- Structure, nème bloc



MEMO→ K7

S → disbuf

BADD

Sauve X=Addeb en 07F2,3

F → disbuf

BADD

Sauve X=Afinen 07F4,5

INIT PIA

FF→ T, FF→ T } debut transmiss

RECYCL

53→ T, 31→ T } debut bloc

((FF-LSB Addeb)_
CODFIN
...x0F)+03=Caract.Intervale
sauve Caract. en 07F1 et T
Addeb (MSB)→ T*
Addeb (lsb)→ T*

X pointe Prog. a transmettre

TJTRAN
LDA,X→ T*

X=Addfin — oui

INCX
Sauve X en 07f2,3
Test Memo 07f3

16 datas? — non
saut 1er et dernier bloc

FINENR
LDA Memo 07F1
NEGA (check sum)→ T

X=Adfin — oui

comparer MSB de Ad en cours & Adfin

adr en cours <Adr.fin — oui

comparer MSB de Ad en cours & Adfin

Plus de 16 cases Memo a transmettre ? — oui

DEBUT CHARGEMnt

53 → T
31 → T

LSB Adfin→ A

FIN CHARGEMENT

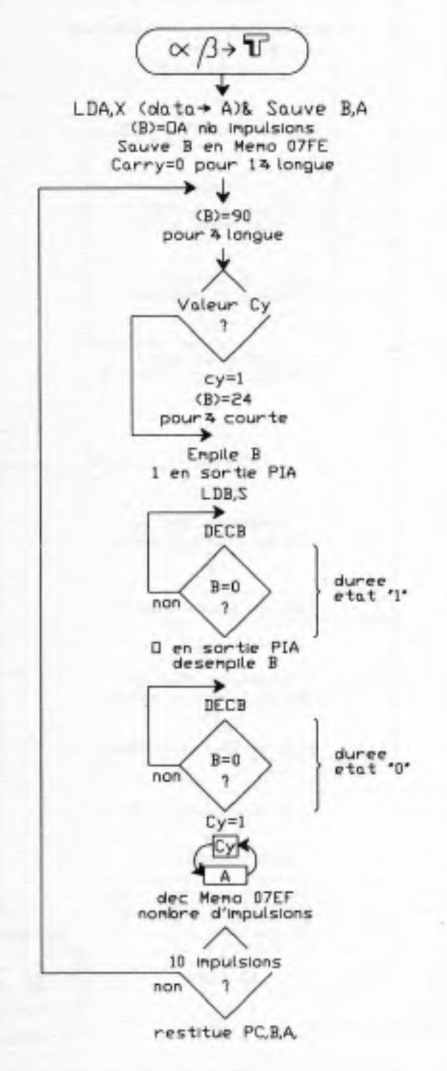
53 → T
4A → T

Affichage F

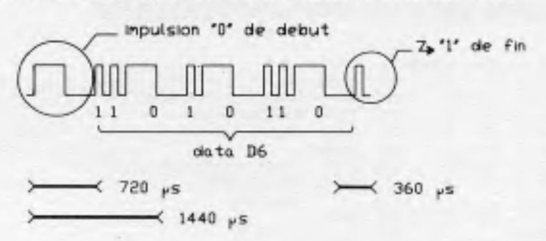
Transmission d'un caractère

Les différents bits constitutifs d'une valeur hexadécimale sont transmis sur la sortie 6 du port B du PIA selon un codage MIL grâce au sous-programme TRANSM. Ce s.p. a été noté $\alpha\beta\rightarrow$ dans l'organigramme général EXLOAD.

```
E6D8 A6   84       A TRANSM LDA    ,X
E6DA 34   06       A DLY5MS PSHS   B,A
E6DC C6   0A       A        LDB    #$0A
E6DE D7   EF       A        STB    <SAVRES
E6E0 1C   FE       A        ANDCC  #$FE     METTRE CARRY A 0
E6E2 C6   90       A BOUCL3 LDB    #$90     DELAI = 720MICROSEC
E6E4 24   02  E6EB          BCC    SAUT
E6E6 C6   24       A        LDB    #$24     DELAI = 180MICROSEC
E6E8 34   04       A SAUT   PSHS   B
E6EA C6   40       A        LDB    #$40
E6EC F7   A005     A        STB    DISCNT   PB6 = 1
E6EF E6   E4       A        LDB    ,S
E6F1 5A              BOUCL1 DECB            DELAI 180 OU 720MICROSEC
E6F2 26   FD  E6F1          BNE    BOUCL1
E6F4 F7   A005     A        STB    DISCNT   PB6 = 0
E6F7 35   04       A        PULS   B
E6F9 5A              BOUCL2 DECB            NOUVEAU DELAI 180 OU 720MICROSEC
E6FA 26   FD  E6F9          BNE    BOUCL2
E6FC 1A   01       A        ORCC   #$01     METTRE CARRY A 1
E6FE 46                     RORA
E6FF 0A   EF       A        DEC    <SAVRES
E701 26   DF  E6E2          BNE    BOUCL3   TERMINER DELAI DE 5MS
E703 35   86       A        PULS   PC,B,A
```



• Ex : transmission de la valeur hexadécimale D6.



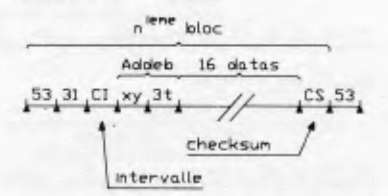
Chargement d'un programme provenant d'un cassette

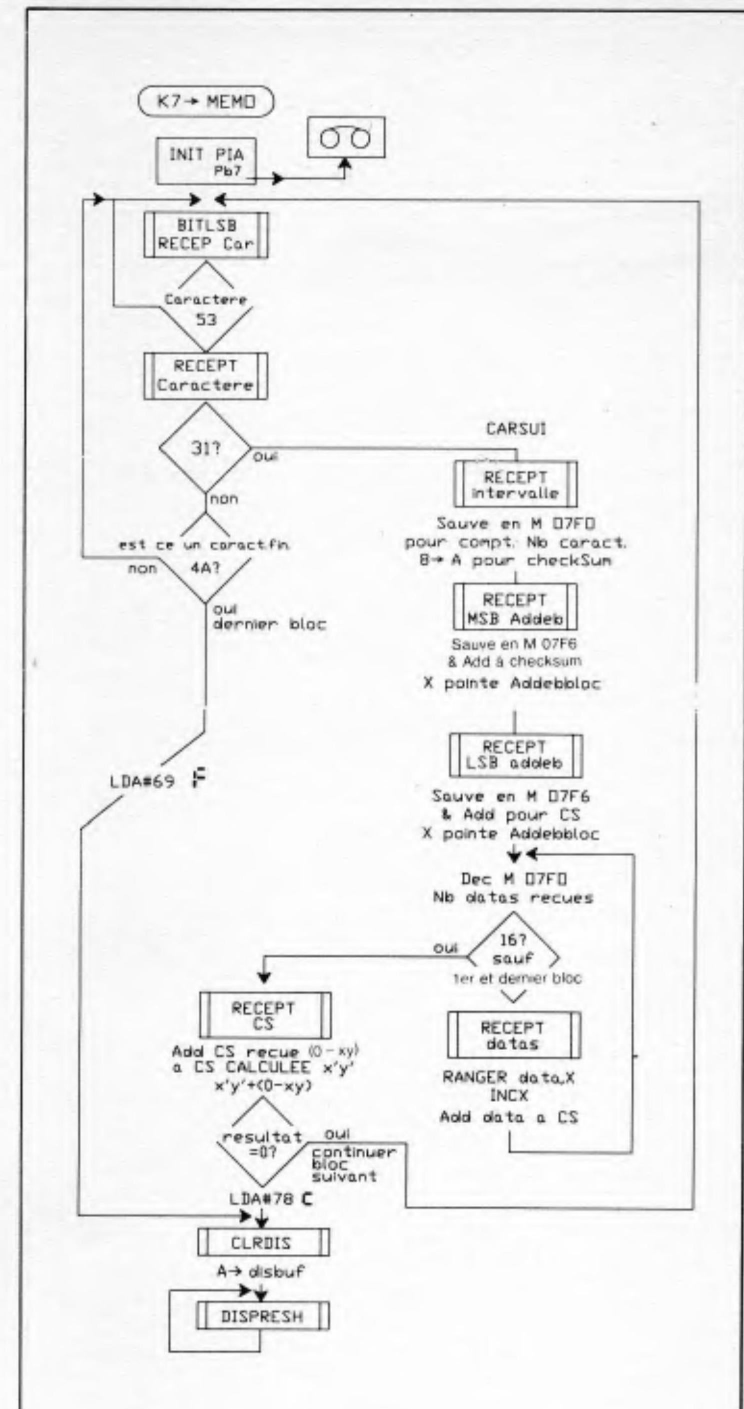
```
E5D9 CC   0000     A EXLOAD LDD    #$0000
E5DC FD   A006     A        STD    SCNREG   ACCES A DDRAB
E5DF CC   FF7F     A        LDD    #$FF7F
E5E2 FD   A004     A        STD    DISREG   PA ET PB EN SORTIE
E5E5 CC   0404     A        LDD    #$0404   PB7 EN ENTREE
E5E8 FD   A006     A        STD    SCNREG   ACCES A DRAB
E5EB CC   FF04     A        LDD    #$FF04   ETEINDRE LES AFFICHEURS
E5EE FD   A004     A        STD    DISREG   ET SELECTIONNER LE 1ier DIGIT
E5F1 8D   D0  E5C3 DETECS BSR    BITLSB   DETECTE CARACTERE DEBUT CHARGEMENT
E5F3 C1   53       A        CMPB   #$53     CARACTERE S TRANSMIS ?
E5F5 26   FA  E5F1          BNE    DETECS   NON,CONTINUER A CHERCHER CARACTERE
E5F7 8D   CA  E5C3          BSR    BITLSB   OUI, CARACTERES SUIVANTS
E5F9 C1   31       A        CMPB   #$31     CARACTERE 1 TRANSMIS ?
E5FB 27   08  E605          BEQ    CARSUI   OUI, CARACTERES SUIVANTS
E5FD C1   4A       A        CMPB   #$4A     NON,CARACTERE FIN = J ?
E5FF 26   F0  E5F1          BNE    DETECS   NON,DETECTER CARACTERE DE FIN
E601 86   69       A AFICHA LDA    #$69     OUI,AFFICHER FIN DU CHARGEMENT
E603 20   32  E637          BRA    DISFIN
```

Mise en mémoire, pointée par X des caractères transmis

```
E605 8D   BC  E5C3 CARSUI BSR    BITLSB   CONVERSION INTERVALLE
E607 D7   F0       A        STB    <SAVCNT
E609 96   F0       A        LDA    <SAVCNT
E60B 8D   B6  E5C3          BSR    BITLSB
E60D D7   F6       A        STB    <SAVB1
E60F 34   04       A        PSHS   B
E611 AB   E0       A        ADDA   ,S+      A + B DANS ACCA
E613 0A   F0       A        DEC    <SAVCNT
E615 8D   AC  E5C3          BSR    BITLSB
E617 D7   F7       A        STB    <SAVB2
E619 34   04       A        PSHS   B
E61B AB   E0       A        ADDA   ,S+      A + B DANS ACCA
E61D 9E   F6       A        LDX    <SAVB1   X CONTIENT ADRESSE DE CHARGEMENT
E61F 0A   F0       A SUICHA DEC    <SAVCNT
E621 27   0A  E62D          BEQ    DERADD   DERNIERE ADRESSE ?
E623 8D   9E  E5C3          BSR    BITLSB   NON, CONTINUER A CHARGER
E625 E7   80       A        STB    ,X+      ET A METTRE EN MEMOIRE
E627 34   04       A        PSHS   B
E629 AB   E0       A        ADDA   ,S+      A + B DANS ACCA
E62B 20   F2  E61F          BRA    SUICHA
E62D 8D   94  E5C3 DERADD BSR    BITLSB   OUI, DERNIERE ADRESSE
E62F 34   04       A        PSHS   B
E631 AB   E0       A        ADDA   ,S+      ERREUR DANS LA TRANSMISSION
E633 27   BC  E5F1          BEQ    DETECS
E635 86   78       A        LDA    #$78     OUI, AFFICHER L'ERREUR
E637 17   FC4E E288 DISFIN LBSR   CLRDIS
E63A 97   FA       A        STA    <DISBUF
E63C 17   FA3C E07B BOUFIN LBSR   DISPRE
E63F 20   FB  E63C          BRA    BOUFIN
```

• Structure nième bloc

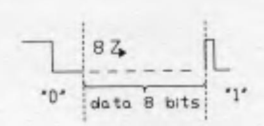


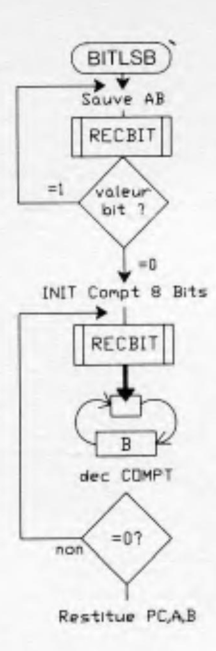


Réception caractère

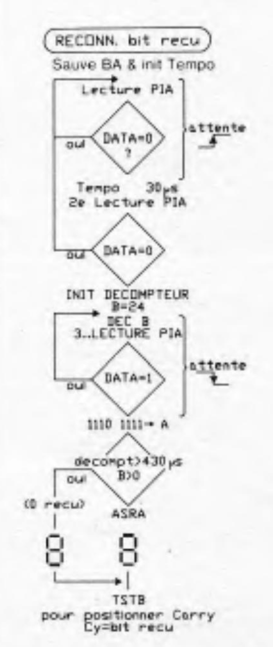
```
E5C3 34   02       A BITLSB PSHS   A
E5C5 8D   D4   E59B BITCA1 BSR    RECBIT   RECONNAISSANCE BIT TRANSMIS
E5C7 2A   FC   E5C5        BPL    BITCA1   BIT CARAC. = 1
E5C9 86   08       A        LDA    #$08     RECONNAISSANCE BITS SUIVANTS
E5CB 8D   CE   E59B COMROT BSR    RECBIT   BIT CARAC=0 RECON BITS SUIVANTS
E5CD 1C   FE       A        ANDCC  #$FE     CARRY=0
E5CF 2B   02   E5D3        BMI    BITCA0   BIT CARACTERE = 0 ?
E5D1 1A   01       A        ORCC   #$01     NON, METTRE CARRY A 1
E5D3 56             BITCA0 RORB            OUI,DEPLACER LA CARRY
E5D4 4A                     DECA            PAR ROTATIONS SUCCESSIVES
E5D5 26   F4   E5CB        BNE    COMROT   CARRY DANS LSB DE ACCB
E5D7 35   82       A        PULS   PC,A
```

- Structure caractere






```
E598 34   06       A RECBIT PSHS   B,A
E59D C6   06       A NOUDLY LDB    #$06
E59F B6   A005     A SCARRY LDA    DISCNT   CHARGER PB7 PIA
E5A2 48                     LSLA            PB7=1?
E5A3 24   FA   E59F        BCC    SCARRY   NON,ATTENDRE UN 1 DEBUT CARAC.
E5A5 5A             DLY30U DECB            OUI,DELAI # 30MICROSEC
E5A6 26   FD   E5A5        BNE    DLY30U
E5A8 B6   A005     A        LDA    DISCNT   CHARGER DE NOUVEAU PB7
E5AB 48                     LSLA            1 TOUJOURS PRESENT?
E5AC 24   EF   E59D        BCC    NOUDLY   NON,NOUVELLE ATTENTE D'UN CARAC.
E5AE C6   24       A        LDB    #$24     OUI,DELAI 430MICROSEC MINI.
E5B0 5A             CARRY1 DECB
E5B1 B6   A005     A        LDA    DISCNT
E5B4 48                     LSLA            TOUJOURS 1?
E5B5 25   F9   E5B0        BCS    CARRY1   OUI,DECOMPTER
E5B7 86   EF       A        LDA    #$EF     NON,DECOMPTE>720MICROSEC.
E5B9 5D                     TSTB            BIT CARAC.=0?
E5BA 2B   01   E5BD        BMI    AFSIGN   OUI,AFFICHER SIGNE POUR 0
E5BC 47                     ASRA            NON,AFFICHER SIGNE POUR 1
E5BD B7   A004     A AFSIGN STA    DISREG
E5C0 5D                     TSTB
E5C1 35   86       A        PULS   PC,B,A
```



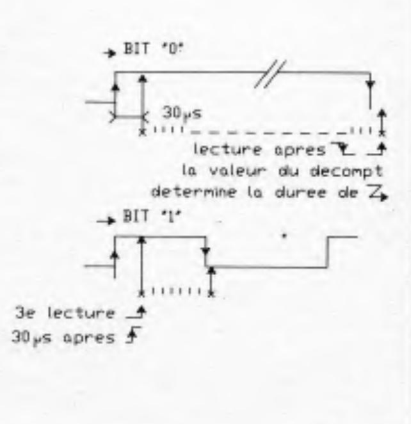


Table des matières